UNIVERSITY OF CALIFORNIA
SANTA CRUZ

Prefetching Complex Access Patterns with Deep Learning

A thesis submitted in partial satisfaction
of the requirements for the degree of

MASTER OF SCIENCE
in
COMPUTER SCIENCE

by

Peter Braun
June 2023

The thesis of Peter Braun is approved:

Professor Heiner Litz, Chair

Professor Jose Renau

Professor Scott Beamer

Peter Biehl
Vice Provost and Dean of Graduate Studies

Contents

List of Figures

List of Tables

Abstract

Prefetching Complex Access Patterns with Deep Learning

Peter Braun

The Von Neumann bottleneck is a persistent problem in computer architecture, causing stalls and wasted CPU cycles. The Von 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 an understanding of what type of memory access patterns an LSTM neural network can learn by studying its effectiveness on a suite of microbenchmarks with well-characterized memory access patterns, and perform a parameter sensitivity analysis to identify the most important model parameters.

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.

Chapter 1

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. 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 [13] 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 [2].

Previous work [6] 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 [7], 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-depth 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 sufficiently generous 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.

Chapter 2

Background

2.1 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, 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 the latency reduction can be up to 100x. Most popular prefetchers such as GHB [13] and stride [4] use recent memory accesses to predict future accesses. These hardware prefetchers generally follow simple heuristics or algorithms. A typical PC-indexed stride prefetcher can identify a fixed number of strided data streams, e.g. 16. In the case a workload concurrently utilizes a greater number of streams, prefetcher performance starts to decline as the many streams compete for resources such as memory access history tables. This sets 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, they 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.

2.2 Deep Neural Networks

Recent advances in deep neural networks have driven their success in fields such as natural 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 [10].

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 the N most recent words, predict the subsequent word. In NLP sequence prediction, there are complex patterns and interrelationships between words in the sentence, similar to the relations between the memory accesses emitted by an application.

2.2.1 Long Short-Term Memory Network (LSTM)

The long short-term memory network (LSTM) [12] has been shown to be effective for this problem [6]. The LSTM is a type of recurrent neural net, which are

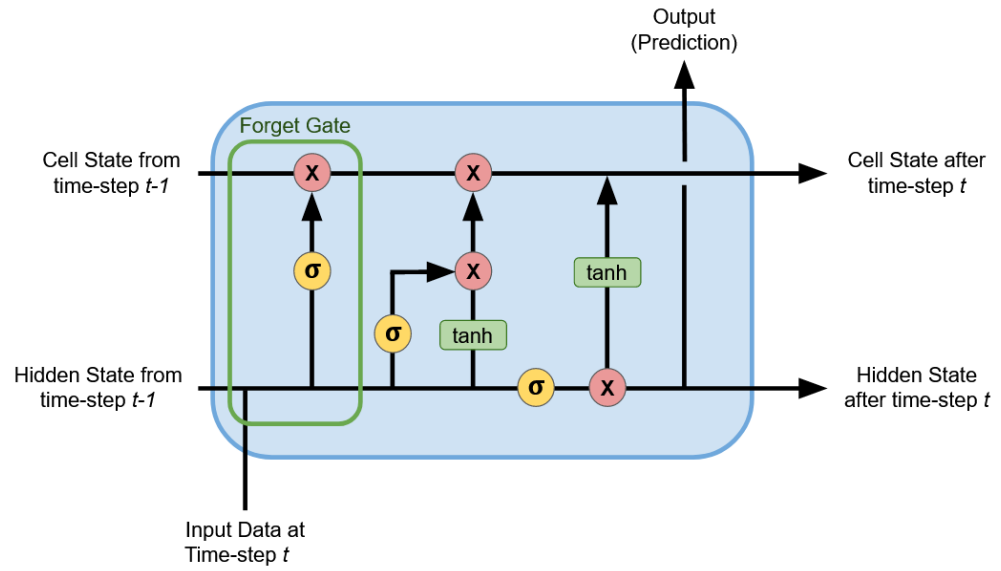

Figure 2.1: A schematic of the LSTM network, showing the inputs, outputs, and computation. Inputs from previous time-steps may influence the current prediction through the cell state and hidden state.

designed for time-series sequences and provide the benefit 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, where the number of layers is referred to as the depth of the network. Each layer has a width specified by a tensor that can be tuned by the model developer. Finally, LSTMs define a *lookback* hyperparameter. Within the scope of this work, this specifies the number of past accesses the model considers to predict the next at each time-step.

The scaling of the number of tunable parameters versus layer size is shown for a single layer LSTM in Figure 2.2. For example, the 8-cell single-layer LSTM contains 2338 parameters.

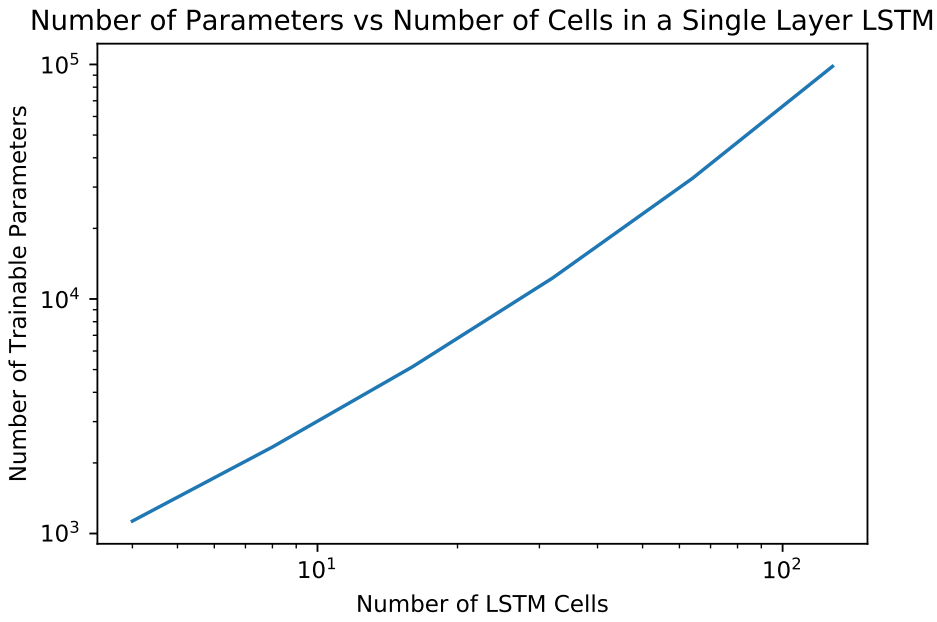

Figure 2.2: The number of trainable parameters scales with the number of LSTM cells. This is shown for a single-layer LSTM model plus a fully-connected output layer that outputs a vector of length 50. The number of parameters ranges from 1K for a 4-cell layer to 98K for a 128-cell layer

Chapter 3

Characterization Framework

3.1 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 state-of-the-art prefetchers are the sequence of the N most recent memory addresses as well as their associated instructions defined by the program counter (PC) 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.

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, creating a very large search space.

A second option is to use memory address deltas. Memory address deltas are computed as the difference between two consecutive memory addresses. Typical memory access patterns such as array or immutable list traversals contain far

fewer distinct deltas than distinct memory addresses. For the same reason, contemporary stride and GHB prefetchers also exploit this characteristic to increase prediction accuracy. Memory address deltas are used for following experiments.

Since the LSTM has been successfully applied for sequence prediction problems such as next-work prediction in natural language, it presents an interesting algorithm to evaluate for data access sequence prediction.

The goals are to:

1. Demonstrate the LSTM's level of effectiveness for memory access sequence prediction, for a set of microbenchmarks of variable complexity.

2. Identify important LSTM parameters and their ideal values through a parameter sensitivity analysis.

3.2 Microbenchmarks

To understand a prefetcher's effectiveness for each class of memory access patterns, we evaluate on a suite of configurable microbenchmarks that demonstrate the major patterns of memory accesses.

To the best of our knowledge this does not exist, so to that end we designed 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:

1. Array traversal (sequential memory accesses)

2. Array of structs (strided memory access with configurable distance)

3. Traversal of a fixed length immutable linked-list (periodic accesses)

4. Compositions of multiple access patterns

Each of these 4 applications generate a regular memory access pattern. The strided patterns continually accessed a memory location that was n bytes away. Since each cache line was 64 bytes, 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. These 4 basic applications were then composed into complex memory access streams by interleaving.

3.3 Trace Generation and Simulation

To obtain the input for model training, we execute the microbenchmarks and obtain memory access traces via DynamoRIO's memtrace [3]. 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 precisely replay the instructions and memory accesses executed by the program. We then simulate using the zsim [15] microarchitectural simulator to perform cache simulation. Cache parameters are described in Table 3.1. The simulation provides the sequence of all L1 cache accesses and L3 misses. The L3 misses are the memory accesses that are targeted for prefetching, while the L1 accesses are the full memory access trace, which provides information that can be used to predict L3 miss addresses. 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 being used to train a DNN.

Table 3.1: Simulation parameters

Parameter	Value
L1i	64B block: 32KiB, 4-way, 3 cycle latency
L1d	32KiB, 4-way
L2	1MiB 8-way, 4KiB-entry

3.4 Data Preprocessing

As shown by [6], 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 [6] [14]. 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 decreases the state space of values to predict.

3.5 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 hyperparameters including the width of the LSTM layer and the lookback size of the LSTM. We train the model in batches of 64 with the ADAM optimizer. We utilize Tensorflow [1] to describe our model and the Keras CuDNNLSTM layer to perform rapid hyperparameter tuning.

The input/output deltas are limited to a subset of all deltas, so they are represented by classes from 1 to 100 and encoded with one-hot encoding.

Chapter 4

Experimental Results

4.1 Accuracy on Microbenchmarks

The accuracy of LSTM models trained on compositions of the previously described memory access patterns is shown in Figure 4.1. 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 model the traversal of an immutable list, have a periodicity of 5-7.

The LSTM performs well, obtaining close to 100% prediction accuracy. For the interleaved streams, every 40 data accesses the program switches to the next stream. In the first four patterns the interleaving of the different patterns is fixed (e.g. pattern1, pattern2, pattern3, pattern4, pattern1, pattern2, ...) so that pattern2 always follows pattern1, etc. In the case of the 4 periodic (random) microbenchmark, each individual stream/pattern follows 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.

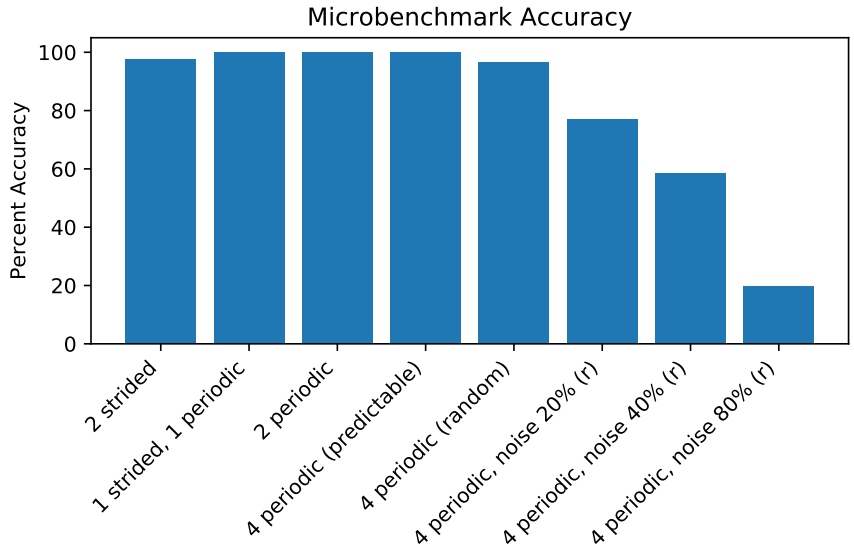

Figure 4.1: Accuracy of LSTM model with microbenchmarks. Each microbenchmark is an interleaving of multiple patterns. The 4 interleaved periodic has a regular predictable switch between patterns (e.g. pattern1, pattern2, pattern3, pattern4, pattern1, ...), and the rest have their next pattern chosen randomly (r). For those a certain percentage of labelled noise is added. Since the 'noise' accesses happen randomly, the DNN understandably is unable to predict them and greater noise leads to lower accuracy. For example, with 80% noise the maximum accuracy we can get is 20% which is indeed what the model obtains.

4.2 Adding Noise

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 into how random noise affects the ability of the LSTM to predict the regular access patterns. As expected, adding these unpredictable accesses causes a drop in accuracy.

To separate noise from signal, we add a separate 'Noise' or 'no predict' class to the model. This enables the model to distinguish noise from predictable accesses. To prevent the model from always predicting 'noise' for noisier traces, the importance of this 'no predict' class was lowered by decreasing its effect on the loss function reducing the class weight.

We also evaluate prediction accuracy for the regular (periodic) accesses only. In particular, we compute accuracy as the fraction of correctly predicted labels of all regular accesses in contrast to all regular and noisy accesses. The accuracy is 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.

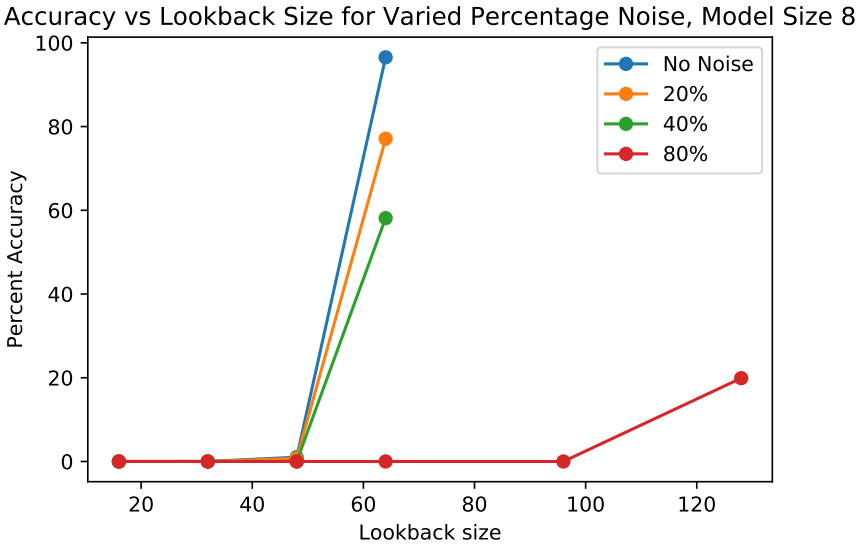

Figure 4.2: Lookback size was found to have a dramatic nonlinear impact on the ability of the model to learn a pattern within the data. There appears to be some threshold lookback size below which the model is just guessing. Above this size, the model rapidly picks up the pattern. The 4 periodic patterns 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.

4.3 Parameter Sensitivity Analysis

4.3.1 Lookback Size

In the following experiments, we analyze how LSTM model complexity affects the prediction accuracy by performing a parameter sensitivity analysis. In particular, we want 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 4.2 we vary lookback size to determine the its impact on a model's accuracy. If the lookback size is too low, the model is unable to learn the pattern. As soon as it reaches some threshold size, the model is able to achieve the full expected accuracy. The threshold for the 0%, 20%, and 40% traces is between 48 and 64. For the 80% trace, the threshold is between 92 and 128 accesses. The pattern is 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. This suggests that the ideal lookback size is related to the window size needed to capture the pattern. For the microbenchmarks thus far, a small model with LSTM layer width of 8 is sufficient to capture the pattern.

To explore limits of this smaller model, we increase the number of unique streams (interleaved patterns). 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 4.3. The threshold for effective lookback size is unaffected by the number of streams as can be seen by the identical trend in both Figures 4.2 and 4.5.

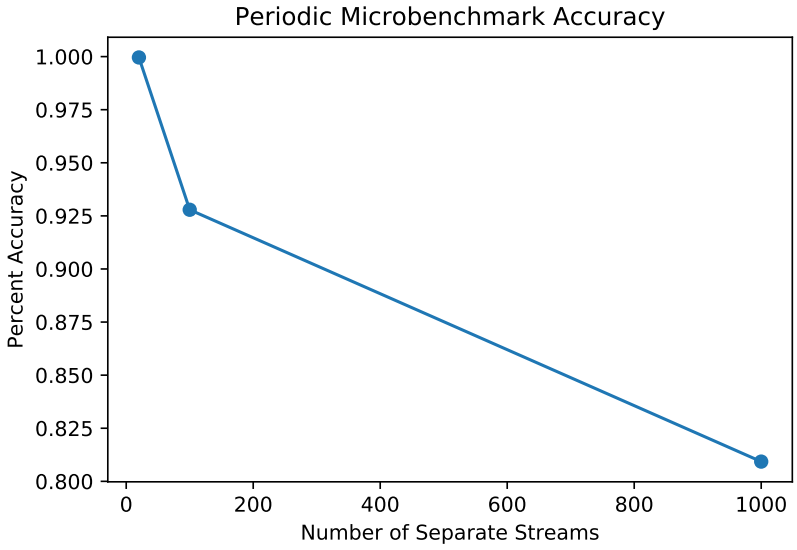

Figure 4.3: Accuracy decreases as the number of interleaved streams (periodic patterns) increases.

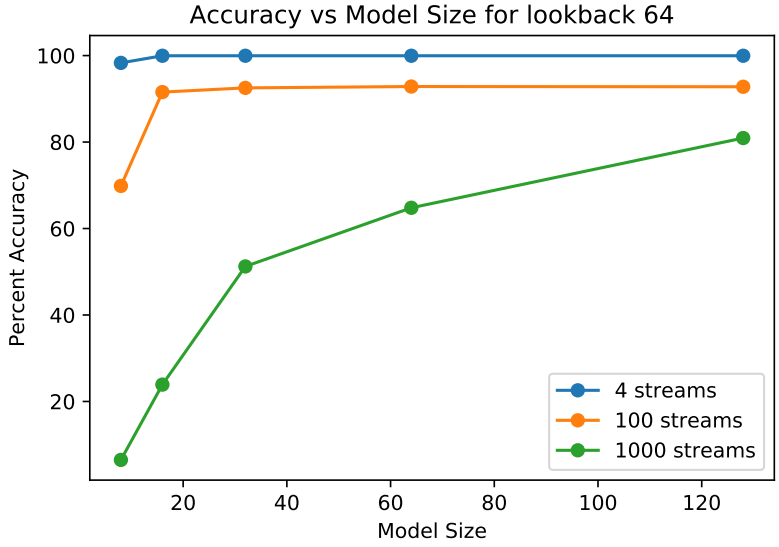

Figure 4.4: Maximum accuracy for interleaved periodic streams increases as the width of the LSTM layer is increased. The importance of larger model becomes much more important as the number of distinct streams to learn is increased.

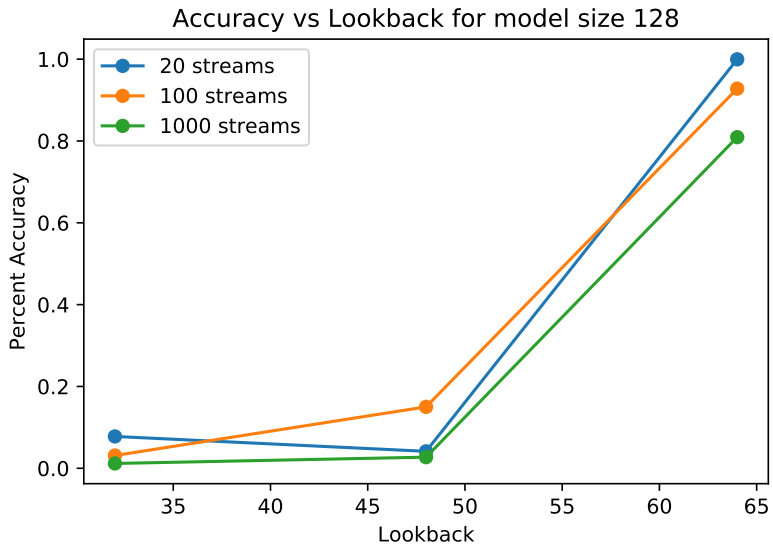

Figure 4.5: For the multiple periodic streams, accuracy dramatically increases once the window size (lookback) is increased past a threshold. The same relationship holds regardless of the number of separate streams.

4.3.2 Model Size

As predicted, the size of the model (number of parameters) becomes important as the information it must learn increases. In Figure 4.4, we observe that a larger model size is required to capture applications with 1000 or more streams. Figure 4.4 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.

4.3.3 Linked List Traversal

For the last microbenchmark, we perform linked list lookups on lists with variable sizes and achieve a maximum accuracy of 99%. Increasing lookback from 32 to 64 only increases the accuracy by about 10 percentage points as seen in Figure

4.6. The dramatic "threshold" observed for the periodic microbenchmarks is not seen here. The number of deltas required to achieve near-complete coverage is much smaller than the number of nodes, probably because nodes may have been allocated adjacently on cache lines. This makes 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.

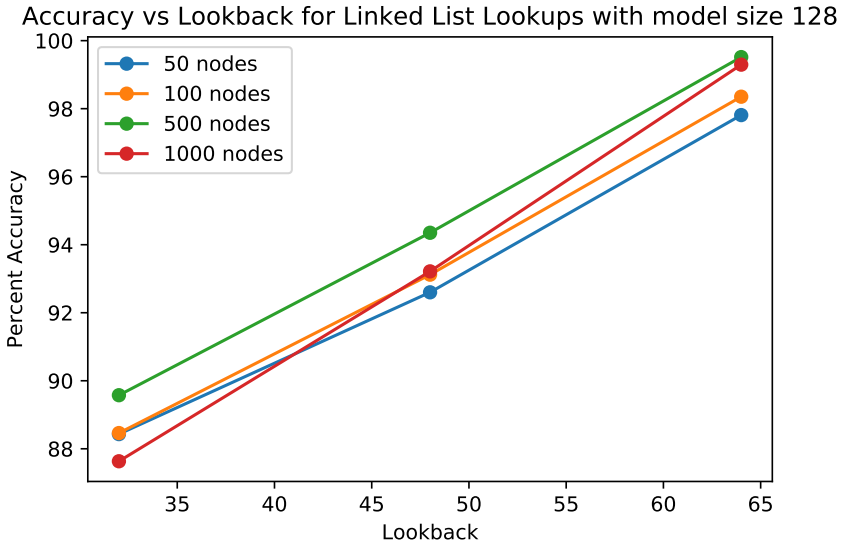

Figure 4.6: Higher lookback captures the local pattern and is able to provide higher accuracy. There is not a significant difference between linked list sizes since a small number of deltas provides almost complete coverage.

Chapter 5

Related Work

Here we highlight several threads of research of machine learning applied to computer architecture. Prior work has used a perceptron to predict whether a branch is taken or not taken [8]. The perceptron learns online 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 [11]. Other research 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 [16]. Another learning algorithm addresses the existence problem in a multiple-WBAN environment using a naive Bayesian classifier [5]. 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 [9], a hardware accelerator for neural network training and inference. Another direction looks at eliminating spurious computations during NN prediction [5], using this with specialized hardware to improve speed and efficiency.

Chapter 6

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 compnents 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.

Bibliography

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mane, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viegas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems, Mar. 2016. arXiv:1603.04467 [cs].

[2] G. Ayers, J. H. Ahn, C. Kozyrakis, and P. Ranganathan. Memory hierarchy for web search. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 643–656. IEEE, 2018.

[3] D. Bruening, T. Garnett, and S. Amarasinghe. An infrastructure for adaptive dynamic optimization. In *International Symposium on Code Generation and Optimization*, 2003.

[4] T.-F. Chen and J.-L. Baer. Effective hardware-based data prefetching for high-performance processors. *IEEE Transactions on Computers*, 44(5):609–623, 1995.

[5] Y. Han, Z. Jin, J. Cho, and T.-S. Kim. A prediction algorithm for coexistence problem in multiple WBANs environment. In *Proceedings of the 8th International Conference on Ubiquitous Information Management and Communication*, ICUIMC '14, pages 1–7, New York, NY, USA, Jan. 2014. Association for Computing Machinery.

[6] M. Hashemi, K. Swersky, J. A. Smith, G. Ayers, H. Litz, J. Chang, C. Kozyrakis, and P. Ranganathan. Learning memory access patterns. *arXiv preprint arXiv:1803.02329*, 2018.

[7] J. L. Henning. Spec cpu2006 benchmark descriptions. *SIGARCH Computer Architecture News*, 2006.

[8] D. Jimenez and C. Lin. Dynamic branch prediction with perceptrons. In *Proceedings HPCA Seventh International Symposium on High-Performance Computer Architecture*, pages 197–206, Monterrey, Mexico, 2001. IEEE Comput. Soc.

[9] N. Jouppi. Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers. In *[1990] Proceedings. The 17th Annual International Symposium on Computer Architecture*, pages 364–373, May 1990.

[10] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, May 2015. Number: 7553 Publisher: Nature Publishing Group.

[11] B. C. Lee and D. M. Brooks. Accurate and efficient regression modeling for microarchitectural performance and power prediction. *ACM SIGOPS Operating Systems Review*, 40(5):185–194, Oct. 2006.

[12] J. Li, X. Chen, E. Hovy, and D. Jurafsky. Visualizing and Understanding Neural Models in NLP. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 681–691, San Diego, California, June 2016. Association for Computational Linguistics.

[13] K. Nesbit and J. Smith. Data Cache Prefetching Using a Global History Buffer. In *10th International Symposium on High Performance Computer Architecture (HPCA'04)*, pages 96–96, Feb. 2004. ISSN: 1530-0897.

[14] L. Peled, S. Mannor, U. Weiser, and Y. Etsion. Semantic locality and context-based prefetching using reinforcement learning. In *Proceedings of the 42nd Annual International Symposium on Computer Architecture*, ISCA '15, pages 285–297, New York, NY, USA, June 2015. Association for Computing Machinery.

[15] D. Sanchez and C. Kozyrakis. Zsim: Fast and accurate microarchitectural simulation of thousand-core systems. In *International Symposium on Computer Architecture*, 2013.

[16] W.-T. Wu and A. Louri. A Methodology for Cognitive NoC Design. *IEEE Computer Architecture Letters*, 15(1):1–4, Jan. 2016. Conference Name: IEEE Computer Architecture Letters.