2018 1st Workshop on Energy Efficient Machine Learning and Cognitive Computing for Embedded Applications

Event Prediction in Processors using Deep Temporal Models

Tharindu Mathew, Aswin Raghavan, Sek Chai
SRI International, Princeton, NJ

1. ABSTRACT

In order to achieve high processing efficiencies, next generation computer architecture designs need an effective Artificial Intelligence (AI)-framework to learn large-scale processor interactions. In this short paper, we present Deep Temporal Models (DTMs) that offer effective and scalable time-series representations to addresses key challenges for learning processor data: high data rate, cyclic patterns, and high dimensionality. We present our approach using DTMs to learn and predict processor events. We show comparisons using these learning models with promising initial simulation results.

2. INTRODUCTION

Large-scale cognitive AI systems have been successfully deployed for a number of applications from image recognition to natural language processing. Such advancements are fueled by availability of training data and a high-performance computing infrastructure. At the same time, there has been much attention drawn in the computer architecture community on accelerating machine-learning processing to support the growth in cognitive AI systems. However, there is still limited research conducted to utilize the power of AI to design better computer architectures and systems.

Today's modern processor designs are based on reactive principles. For each new workload, a new benchmark is created, execution profiles are created, and a new HW accelerator is built solving merely near-term and a-priori known performance issues. This process causes unacceptable stresses on other system components, including power delivery network, clock synchronization, and memory subsystems. The end result is a reactive approach to the design and operation of the processor: see a problem, design a solution, look for the next problem, and repeat. What is needed is a new design paradigm that breaks away from this reactive design cycle to significantly provide orders of magnitude of improvement in power efficiency and computing performance.

Processor hardware performance improvements have tapered off because it is becoming increasingly challenging to scale to lower power consumption. The problem with conventional system design is that it overlooks the interactions that can be foreseen and mitigated by taking a holistic look at the interactions occurring across the entire system. While designers would like to ideally take a holistic picture into account, the challenge lies in manually understanding and rationalizing the interactions at design time. This is simply beyond current capabilities, especially because the system designers cannot fully anticipate runtime activity and the consequential impact on performance, power and reliability.

This short paper presents an AI-enabled approach to computer architecture design. We posit that the fundamental aspect for AI-enabled computing is the development of **efficient, scalable ML representation** that can learn naturally the complex processor behavior and events from processor data. In this paper, we explore the use of Deep Temporal Models (DTMs) that are hierarchical and multi-staged, and have been used successfully in large-scale image processing, with recent extensions to video processing. Current non-temporal models like Convolutional Neural Networks (CNN) treat data as independent and static without the inherently temporal properties. This is inevitably sub-optimal, especially from the point of view of learning. We show DTMs for use in computing and at the same time leverage temporal models that are specialized for data sequences rather than static data. We leverage the generative properties of DTMs to forecast processing and data access needs in forthcoming phases of the computation.

3. BACKGROUND

There have been multiple generations of machine learning (ML) applied to the computing domain. While there is a recent push in research for making deep learning processing efficient [1-5], there is a disproportionate amount of research using deep networks *for* computing. The first generation used Support Vector Machines (SVMs) and shallow networks [6] to encode logic functions, for example for branch prediction and power management. SVMs are discriminative classifiers; they may provide high performance in small problems, however, their complexity increases in large problems. For global processor behaviors with many cyclic patterns, SVM approaches are likely to give poor performance. These machine learning approaches have been used in processor control circuitry, but to the best of our knowledge, they: (1) are very component specific, and thus have limited functionality, (2) do not learn global system behaviors and do not scale well, and (3) use very shallow neural networks, without spatial-temporal learning.

The second generation attempted using modern deep learning techniques. Some work focused on using Reinforcement Learning (RL) for task scheduling [7] and device placement/selection [8]. The main limitation of RL is that the actions of the agents and reward functions should be defined based on the system and task requirements. That is, without a well-premised learnt representation of processor behavior, the RL approach may not be effective or optimal.

Within the AI field, DTMs are a rapidly growing research area driven by the need to study and exploit sequential data. What distinguishes DTMs from their contemporaries is the use of so-called "distributed" representations: explaining data patterns by a set of interacting features instead of a category or prototype. The AI research field has evolved in a number of important ways: 1) the amount of data readily available; 2) more powerful computational resources to leverage that data; and, 3) improved algorithms for training in both supervised and unsupervised learning modes.

The introduction of Deep Belief Networks (DBNs) by Hinton et al. [9] is widely viewed as the breakthrough that stimulated the modern view of Deep Learning. Up to this point, classical multi-layer architectures such as deep neural networks were known in the ML community but were universally accepted as difficult to train in practice. Hinton and colleagues showed that DBNs could be grown incrementally, by composing simple models called Restricted Boltzmann Machines (RBMs). The models were trained in sequence, each model learning to represent the one before it. Once this "greedy" initialization strategy completed, the model could be *fine-tuned* end-to-end. Labels could be introduced in this second stage of learning. By first discovering representations, learning became effective and efficient. DTMs are evolved by extending RBMs to capture temporal dependencies [10, 11].

Sutskever et al. [12,13] have demonstrated progress of memorization problems using Recurrent Neural Networks (RNNs) that have simple, unimodal output distributions that are better suited for predictions tasks. Bengio's group [14,15] has identified and summarized strategies for constructing and optimizing deep RNN models. They have found that clipping gradients, leaky integration, advanced momentum, more powerful output probability models, and sparser gradients generally help generalization performance as well as training performance. With advancement in DTMs (e.g. RBMs, RNN/LSTMs), a number of AI applications have been successful demonstrated: human activity recognition from video [16-18], modeling of 3D volumes [19], and speech processing [20].

None of the prior approaches look at the computing problem from a deep temporal representative-learning point of view, nor address the generative side for predicting expected resource requirements. Our work described in this paper, covers DTMs that handle temporal data in a unified manner, since both local and global information are combined and analyzed as an end-to-end system.

Our key objective is to lay the foundation for a **proactive** computing platform that is adaptive and can manage its own operational needs. The ultimate goal is to eliminate the penalties to system design and operation that arise from the use of incremental circuit techniques and micro-architectural changes, which increases system complexity and cost, and affects power consumption – all for the sake of performance over a generalized set of benchmark suites that may have little in common with the computations that will actually be executed on the machine. More specifically, hardware designers tend to overdesign to provide a minimum performance guarantee for software, and this tiered approach between hardware and software has driven processor architectures to be complex, rigid and over-designed. For AI-enabled design to be successful, we advocate an approach that eliminates system overheads, having resource controlled by a data-driven AI-engine.

4. DEEP TEMPORAL MODELS

We are developing DTMs to learn and predict data-driven micro-architectural behaviors to achieve an optimal (or near-optimal) execution flow. The goal is to learn complex interactions between processor events that have large variations across different execution phases. We train the DTMs using a training dataset comprising of recorded and simulated performance statistics during processor operation using an architecture simulator. We want to provide a method where the AI-aspects are not overdesigned and scale across different workloads. In the future, we can extend the training using adversarial and reinforcement learning approaches to capture events not previously observed or characterized in a benchmark suite. At runtime, the trained DTMs will recognize patterns of micro-architectural events, and will infer predictions on future behavior of the workload. An effective prediction must meet two criteria: 1) it must anticipate a future behavior accurately to prevent un-optimized or unnecessary hardware operation. 2) it must predict a new execution phase or resource need with enough lead time to be able to initiate (re)configuration without throttling platform operation.

DTMs recognize phases of program execution based on signatures of code sequences and processor states. The generative model allows us to reconstruct signals and patterns through inference, so in essence, we can predict future processor behavior. Much like how gesture recognition is trained on motion caption data [21], we are capturing multi-dimensional data from processor operation, and using that data to train our network.

In this paper, we demonstrate the use of DTMs to predict aspects of processor operations based on spatial-temporal analysis, extending our paper using a new variant of the Conditional Restricted Boltzmann Machines (CRBMs) [22], with Long-Short-Term Memory (LSTM). Due to space constraints, we will present LSTM results in this paper.

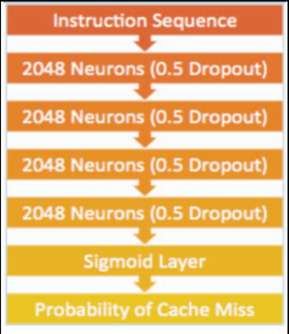
Instruction Sequence
2048 Neurons (0.5 Dropout)
2048 Neurons (0.5 Dropout)
2048 Neurons (0.5 Dropout)
2048 Neurons (0.5 Dropout)
Sigmoid Layer
Probability of Cache Miss
Figure 1. LSTM Deep Temporal Model

Figure 1 shows the LSTM deep learning model we used to model GPU processor activity as a time-series problem. The GPU instruction sequences (e.g. add, mov, call) are m dimensional inputs with the occurrence of a cache miss as the binary output at each execution cycle. The LSTM model is then trained to look at a sequence of instructions, and predict the occurrence of a cache miss, n cycles ahead. The LSTM model uses 2048-neuron LSTM [23] cell in 4 successive layers. To improve regularization, a drop out ratio of 0.5 is applied after each layer. A sigmoid layer outputs the probability of a cache miss. A threshold of 0.5 is applied to create a binary output that represents the cache miss.

In our experiments we consider each workload (each with ~20,000 instructions) of the Rodinia Benchmark, and stack five of these together to form the training set. The training converges in usually 2000 epochs using Adagrad, consuming 1 hour on a Nvidia Titan X GPU. The testing is done on a workload by predicting n cycles ahead. This completes in <1 minute.

5. Experiments

We designed experiments to explore and understand the complexity of deep learning algorithms required to learn processor (CPU/GPU) operations. We used the Rodinia Benchmark [24] software suite, which consists of a number of compute intensive kernels used by researchers to characterize system performance. We setup a GPU simulator [25] to run the software suite, and we trained our algorithms to predict processor events. Table 1 shows the twelve (12) programs of the Rodinia Benchmark selected to run through the GPU simulator simulating a NVidia GTX 480. Executed instructions across cores and data cache read miss (DC_RM) data are extracted from all of the programs.

Figure 2 shows the results of learning workloads from one set of benchmarks to apply to another set of benchmarks. Our learning algorithms are set to predict cache misses as an indication of how we can predict processor operational state. We show that deep learning models (LSTM) can learn temporal sequences and can outperform regular machine learning algorithms (SVM). In addition, LSTM can outperform an approach (naïve baseline) that randomly predicts. For the Baseline, we use the true label ratio of the training set as the prediction probability. We preprocessed data for the SVM in two ways. The first method included time as a feature, by stacking a set of features and augmenting the data where each input consists of a set of instruction sequences spanning the number of cycles. The second method fed the data directly to the SVM without augmentation. It was seen that the time-based augmentation of the data did not give any performance benefit, and exponentially increased the training set size, increasing training time. We omit the results of the first method and present comparisons of the second method for brevity.

In these experiments, we chose n = 32, to predict cache misses 32 cycles ahead from a given cycle. This window can be set during training, and we have shown in [22] for windows as large as 128 cycles. Larger window sizes would require more memory storage in the LSTM network. Five (5) randomly picked training programs are used as training programs to predict misses of an unseen test program. Experiments were conducted with n in the range of [1..128] with similar behavior as observed in figure 2. The CFD Solver (C) performs well with the random predictions of the baseline giving us an outlier in the benchmark. We attribute to the unstructured grid and the random behavior of fluid dynamics workload. Each workload has ~20,000 instruction sequences on average. This results in a sampling size of ~20,000 sequences per workload.

One salient result is the demonstration that DTMs can generalize what it has learned to apply from one workload to another. Our results show that compute heavy workloads may have common patterns observable through the histogram of executed instructions, and that these patterns can be learned. To further improve performance, we can also use memory addresses and other processor information as inputs to the DTMs.

There are several takeaway messages in Figure 2. First, the intent of Figure 2 is to show how DTMs can learn to predict cache misses from instruction sequences alone. There are many other inputs to DTM that can be considered to improve prediction accuracy. Second, we show how DTMs can generalize to learn from different benchmarks. Specifically, we can treat DTMs as a tool to craft new processor control strategies by providing the right training data. As such, we break away from the reactive design principles that plague current design methodology. Third, we show that using a data driven method, designers can then target smaller subdomains rather than having a generalized solution for all

	Application	Dwarf	Domain	Problem Sizes
1	Back Propagation (BP)	Unstructured Grid	Pattern Recognition	65536 input nodes
2	Breadth First Search (BF)	Graph Traversal	Graph Algorithms	1000000 nodes
3	CFD Solver (C)	Unstructured Grid	Fluid Dynamics	97k elements
4	Heart Wall Tracking (HW)	Structured Grid	Medical Imaging	609 x 590 pixels/frame
5	HotSpot (HP)	Structured Grid	Physics Simulation	500 x 500 data points
6	Kmeans (K)	Dense Linear Algebra	Data Mining	204800 data points, 34 features
7	Leukocyte Tracking (LK)	Structured Grid	Medical Imaging	219 x 640 pixels/frame
8	LU Decomposition (LU)	Dense Linear Algebra	Linear Algebra	256 x 256 data points
9	MUMmer (M)	Graph Traversal	Bioinformatics	50000 25-character queries
10	Needleman-Wensch (N)	Dynamic Programming	Bioinformatics	2048 x 2048 data points
11	SRAD (SR)	Structured Grid	Image Processing	512 x 512 data points
12	Stream Cluster (SC)	Dense Linear Algebra	Data Mining	65536 points, 256 dimensions

Table 1. The Rodinia workload type and problem space

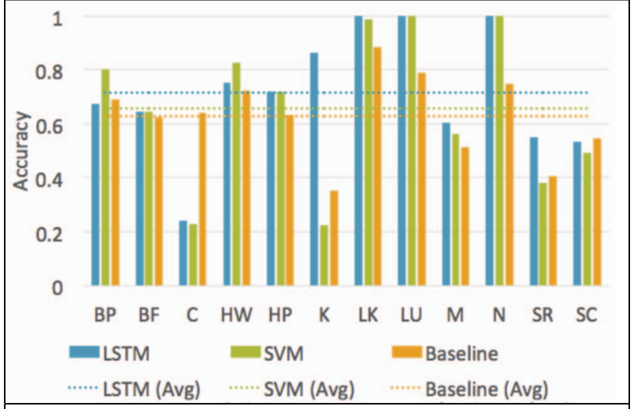

Figure 2. Generalized DTM learning on five random workloads (LSTM vs. SVM) in predicting data cache misses. (Training data from Rodinia Benchmark -- BP, BF...SC).

benchmarks. That is, several cache policies can be learned independently and loaded into hardware at runtime. In this manner, designers can move away from overdesigning complex control mechanisms to cover all possible scenarios.

Similarly, in Figure 3, we show that LSTM can outperform SVM with less data samples (e.g. uses less successive training data to achieve performance). Here, we show cross-dataset prediction where multiple program instructions are stacked successively for training. e.g. numbered 1-6 (from Table 1) using training data from BP+BF+C+HW+HP+K that are stacked together. Each workload has ~20,000 instruction sequences on average. We show results for two benchmarks: Back Propagation (top) and Stream Cluster (bottom).

For this evaluation, we use the F1 score for evaluation. We have determined that training validation needs proper evaluation criteria [22]. For example, *model-accuracy* criteria can be misleading because cache miss events are rare (e.g., about 10% for instruction cache). Instead, we use the *F1* and *Mathew Correlation Coefficient* (MCC) metrics because they better represent the predictive nature of the DTM, when the observed value of has a low duty cycle of 3.

For Back Propagation benchmark (unrelated to back propagation occurring when training the DTM), we show that the LSTM performance increases and out performs other approaches after two training data sets. In comparison, SVM approaches take more training data and can saturate in performance. This is a good example where the LSTM model is able to learn new features given additional data and improve overall performance. For Stream Cluster, we show that LSTM maintains high performance with increasing training data, while SVM saturates with more data. In this example, the LSTM model is able to use less training data to learn key features, and it is able to retain those learnt features over successive training sequences with more training data.

Our results are currently focused on single processor behavior (e.g. cache misses), and there will be cases, we cannot generalize. For example, it is readily understood that streaming and random data access are inherently different. Our system uses deep temporal models to learn complex processor activities, and we do believe that this approach can be used in many different applications. For example, DTMs can be used to eliminate operational overheads and to allow for user personalization by anticipating power fluctuations, program operation, and system faults to achieve data-driven performance gains.

GPUs have large numbers of processor cores, and activity prediction is complex problem due to the high level of contention among thousands of parallel threads, and data dependencies on input data. It is important to note that we model the GPU processor activity as a time-series problem. Specifically, the GPU instruction sequences (e.g. add, mov, call) are *m* dimensional inputs with the occurrence of a cache miss as the binary output at each execution cycle. As such, our DTMs are learning to look at best sequences of instructions that predict the occurrence of a cache miss, *n* cycles ahead. Furthermore, our DTMs are generative models that learn internal representations of hidden variables and can thus better learn complex data sequences. In comparison, a discriminative model, like the many learning algorithms for perceptrons, only models output target variables, and therefore, they cannot express complex relationships between the observed and target variables. This is analogous to the use of DTMs for natural language processing (NLP) in which words in sentences can be predicted based on proceeding sentence structure.

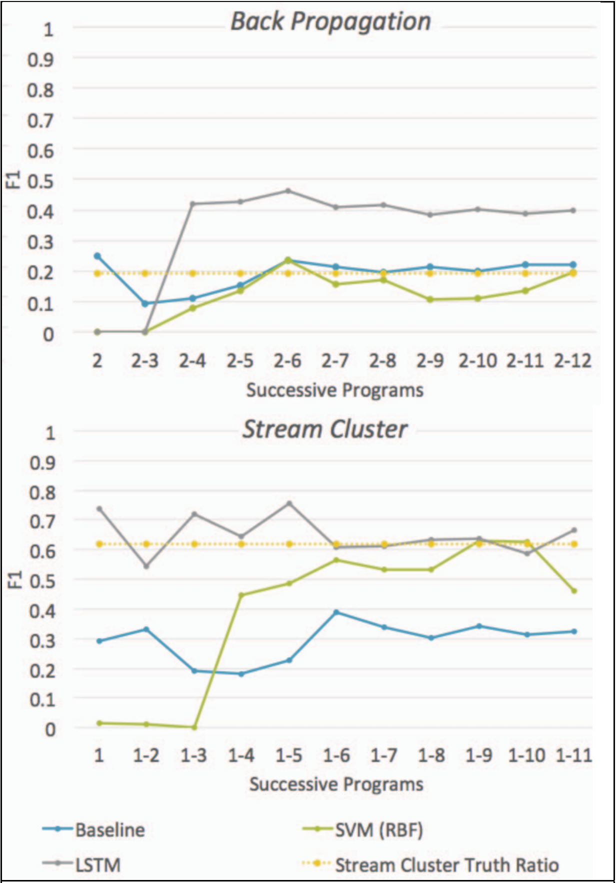

Figure 3. Results showing performance of Deep Learning LSTM using successively more training data.

6. CONCLUSIONS

Deep Temporal Models (DTMs) offer a unified AI-framework to learn and predict micro-architectural events in modern computer architecture. They enable hardware that is adaptive and scalable to support current and workloads in future cognitive systems. In this short paper, we presented initial results using DTMs to learn from training data generated from a computer architecture simulator. Our results show that DTMs can generalize what it has learned to apply from one workload to another. As such, we summarize that through a data-driven methodology, our DTMs has extracted key features and patterns that can infer future processor behaviors. In the next step, we are evaluating low-power hardware architectures and circuits to implement DTMs using low-bit precision [1]. We plan to focus on the reduction in time/energy associated with storage and movement of data.

Using DTMs, we anticipate the following benefits for computer architecture design: 1) Designers no longer codify and implement circuits for branch prediction or other control logic. Instead, DTMs offer a data-driven approach where the designers must collect sufficient training data; 2) Compilers can analyze and identify a mix of instructions that cause performance bottlenecks and schedule them dynamically based on statistical inference; 3) Processor activity can be predicted to dynamically set voltage, current, and other system level parameters. This can be applied to a number of applications including cyber security in the form of anomaly detection. Specifically, the generative DTM learns normal processor behavior, and an active malware will change processor behavior (e.g. unexpected branch or fault/exception). More complex behaviors (e.g. unexpected sequences of instructions or events) can also raise a warning (e.g. a read from memory and transfer of data over the network).

In related future work, we intend to expand our experiments to show system level variability (frequency, memory subsystem, latencies, pipeline width), in addition to workload variability. We have also applied this methodology towards anomaly detection for cyber security related applications.

7. ACKNOWLEDGMENTS

We would like to thank Dr. Mohamed Amer and Dr. Vijay Reddi for their support of part of this effort. This material is based upon work supported by the Office of Naval Research (ONR) under contract N00014-17-C-1011, and NSF #1526399. The opinions, findings and conclusions or recommendations expressed in this material are those of the author and should not necessarily reflect the views of the Office of Naval Research, the Department of Defense or the U.S. Government.

8. REFERENCES

[1] A. Raghavan, et al., "BitNet: Bit-Regularized Deep Neural Networks", arXiv:1708.04788, 2017

[2] M. Courbariaux et al., "Binarized neural networks: training neural networks with weights and activations constrained to +1 or -1," arXiv:1602.02830, 2016.

[3] J. Kung, et al. "Efficient Object Detection Using Binarized Neural Networks," *JSPS*, 2017.

[4] S. Han, et al., "EIE: efficient inference engine on compresses deep neural network" arXiv preprint arXiv:1602:01528, 2016.

[5] N. Jouppi, et al., "In-Datacenter Performance Analysis of a Tensor Processing Unit", ISCA 2017.

[6] J. Leng, et al. "Safe limits on voltage reduction efficiency in GPUs: A direct measurement approach", Proceedings of the 48th International Symposium on Microarchitecture. ACM, 2015.

[7] D. Hanus, *Smart Scheduling: Optimizing Tilera's Process Scheduling via Reinforcement Learning*, MSEE thesis, MIT, Feb 2013

[8] A. Mirhoseini, et al., "Device Placement Optimization with Reinforcement Learning", Arxiv1706.04972,2017

[9] G. E. Hinton, et al. "A fast learning algorithm for deep belief nets" Neural Computation 2006

[10] I. Sutskever and G. E. Hinton, "Learning Multilevel Distributed Representations for High-Dimensional Sequences" AISTATS 2007

[11] R. Memisevic and G. Hinton. "Learning to represent spatial transformations with factored higher-order Boltzmann machines" Neural Computation 2010.

[12] S. Hochreiter, et al. "Gradient flow in recurrent nets: the difficulty of learning long-term dependencies" IEEE F. Guide Dynamical Recurrent Networks, 2001.

[13] I. Sutskever, et al., "On the importance of initialization and momentum in deep learning" ICML 2013

[14] Y. Bengio, et al., "Advances in optimizing recurrent networks" ICASSP 2013

[15] R. Pascanu, et al., "How to construct deep recurrent neural networks", arXiv:1312 2013.

[16] A. Karpathy, et al., "Large-scale Video Classification with Convolutional Neural Networks", CVPR 2014

[17] G. Taylor, et al., "Convolutional learning of spatio-temporal features" ECCV 2010

[18] K. Simonyan and A. Zisserman "Two-stream convolutional networks for action recognition in videos", arXiv:1406.2199, 2014.

[19] S. Ji, et al., "3D Convolutional Neural Networks for Human Action Recognition" PAMI 2013

[20] G. Hinton, et al, "Deep Neural Networks for Acoustic Modeling in Speech Recognition" SPIE 2012.

[21] M. Amer, et al., "Deep multimodal fusion: A hybrid approach", IJCV, Feb. 2017.

[22] A. Raghavan, et al. "GPU Activity Prediction using Representation Learning", ML Systems Workshop, arXiv:1703.09146 (2017)

[23] Hochreiter, Sepp, and Jürgen Schmidhuber. "Long short-term memory." Neural Computation (1997).

[24] Che, et al., "A Characterization of the Rodinia Benchmark Suite with Comparison to Contemporary CMP Workloads", IISWC2010

[25] Bakhoda et al. Analyzing Cuda Workloads Using a Detailed GPU Simulator. ISPASS2009