EAST-DNN: Expediting Architectural SimulaTions Using Deep Neural Networks

ABSTRACT

A rapid and accurate architectural simulator is a cornerstone for an efficient design-space exploration of computing systems. In this paper, we introduce EAST-DNN, a feed-forward deep neural network, to accelerate architectural simulations. EAST-DNN achieves $> 10^6\times$ speedup with an average prediction error of 4.3% over the baseline simulator. It also achieves an average of 2× better accuracy with at least 2.3× speedup compared to state-of-the-art.

CCS CONCEPTS

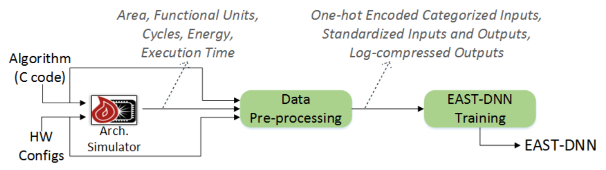
• **Computer systems organization → Computer Architecture**; *Deep Learning*; VLSI.

KEYWORDS

Deep Neural Network, Hardware Accelerators.

1 INTRODUCTION

Application-specific hardware accelerators are proliferating in various computing platforms as they outperform general-purpose architectures [7]. An efficient design of such accelerators is crucial. Thus, it is important to explore the entire design-space of all possible parameters (number of compute units, on-chip memory, operating frequency, etc.). Current state-of-the-art architectural simulators, such as ALADDIN [10], can estimate performance, energy and area (PEA) prior to RTL design with a minimal degree of error, but at the cost of significant time consumption. It is therefore necessary to accelerate the simulation times, which in turn enables a rapid exploration of a large set of design points. Traditional sampling methods can accelerate simulations at the cost of accuracy [3]. Machine-learning methods like random search, combined statistical sampling with active learning (with adaptive boosting) [9] and neural network based learning (with only one hidden layer) [8] can quicken simulations, with relatively low accuracy. In this paper, we introduce EAST-DNN —a feed-forward deep neural network to accelerate architectural simulators. We evaluate the accuracy and speedup of EAST-DNN, using deep-learning accelerator simulations as a case study, against state-of-the-art random forest (RF) [6] and extreme gradient boosting (XGB) [2] (XGB outperforms adaptive boosting [1, 4]). We find that EAST-DNN achieves 2.4× and 2× better accuracy with respect to RF and XGB, respectively. EAST-DNN achieves $> 10^6\times$, 38.9× and 2.3× speedups over baseline simulations, RF and XGB, respectively.



Area, Functional Units, Cycles, Energy, Execution Time
One-hot Encoded Categorized Inputs, Standardized Inputs and Outputs, Log-compressed Outputs
Algorithm (C code)
HW Configs
Arch. Simulator
Data Pre-processing
EAST-DNN Training
EAST-DNN

Figure 1: DNN Acceleration Methodology.

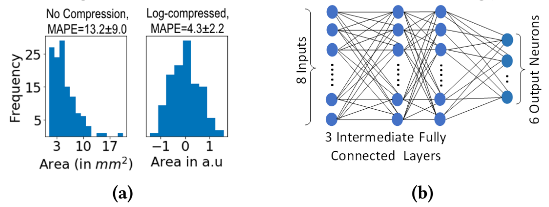


Figure 2: (a) Logarithmic compression and standardization of outputs. (b) Schematic of EAST-DNN.

2 METHODOLOGY

Figure 1 illustrates overall approach. An architectural simulator generates PEA estimates by varying algorithmic and hardware configuration parameters (e.g., Table 1). A feed-forward deep neural network is trained with these configuration parameters as input, and predict PEA estimates as final output. A data pre-processing stage is required to transform data into a format that yields high accuracy of PEA predictions by EAST-DNN (details shown later).

2.1 Case Study: Deep-learning Accelerators

We employ ALADDIN [10] simulator which accepts algorithmic representations (C code) of any application workload and hardware configurations (such as number of memory banks, clock period and hardware pipelining) as input parameters. ALADDIN then estimates power consumption, number of clock cycles, area of hardware accelerator (assuming a certain technology node—45nm in this paper) and the number of functional units (adders, multipliers and registers). Energy and execution time are derived from power, cycles and clock period. Convolution operations constitute the majority of deep learning workloads. Thus the sizes and depths of input feature maps and filters involved in convolutions are considered as algorithmic parameters. The details of parameters for convolution workloads are summarized in Table 1. For hardware configuration parameters, we vary the number of memory banks (M_b), clock period and pipelining (P). For M_b, the ratio *high*:*med*:*low* is maintained at 8:4:1. Value “*high*" represents highest number of memory banks—one bank per functional unit. Pipelining can improve execution time by removing dependencies between functional units.

Table 1: Parameter Values

Workload Parameters	**Value Set**	**HW Parameters**	**Value Set**
Input Feature Width (W)	{10,20,30,40}	Memory Banks (M_b)	{high, med, low}
Input Feature Height (H)	{9,18,45}	Clock Period (in ns)	{1,2,3,4,5,6,10}
Filter Size (w×h)	{7×7}	Pipelining (P)	{yes, no}
Channels (C)	{3}		
# Filters (f)	{8,16,32}		

2.2 EAST-DNN Structure and Pre-processing

We use a feed-forward network to predict PEA estimates. Figure 2.b illustrates EAST-DNN, which consists of 4 fully-connected layers—3 hidden layers of 65, 280, 120 neurons per layer and a final output layer of 6 neurons. We have considered energy, execution time, area, adders, multipliers and registers as output as this leads to faster convergence to accurate model. We apply data transformation mechanisms to raw input and output features before feeding them to EAST-DNN. For input features, we one-hot encode linguistic representations to numerical values, for instance, $\{high, med, low\}$ values of M_b are encoded as {100,010,001}. For output labels, we apply logarithmic data-compression to stabilize the skewed distribution (Figure 2.a), which in turn reduces prediction error. Finally, the one-hot encoded input and log-compressed output features are standardized as follows:

$$\dot{X} = (X - \bar{X})/\sigma(X)$$

where $\bar{X}$ is the mean and $\sigma(X)$ is the standard deviation of X.

2.3 EAST-DNN Training

The loss function of the feed-forward network is defined by Mean Average Percentage Error (MAPE):

$$MAPE = ((Val_{true} - Val_{pred})/Val_{true}) * 100$$

where Val_{true} represents the simulator-output value of energy, time, area or their mean, and Val_{pred} represents the corresponding value predicted by EAST-DNN. For training, EAST-DNN weights are initialized with Glorot normal distribution [5] and regularized by L2 kernel regularization. EAST-DNN is trained using TensorFlow[1] with Adam's optimizer with a learning rate of 0.03, batch size of 40, exponential decay rate of 0.96 and 900 training epochs.

3 SIMULATION RESULTS

The sample space consists of 8 categories, determined by their input image (feature) size fed into ALADDIN simulations. These categories are divided into two sets to illustrate generalization capability of EAST-DNN, namely high-cardinality={10x9x3,20x9x3,40x9x3, 40x18x3, 10x45x3} and low-cardinality={30x9x3,20x45x3,10x8x3} with 5 and 3 categories of input image domains respectively. Each category in high-cardinality subset contains larger number of samples than individual category in low-cardinality subset.

First, we train EAST-DNN with individual categories in high-cardinality. Table 2 summarizes the accuracy of EAST-DNN against RF [6] and XGB [1] implementations[2]. Each category in high-cardinality set consists of 126 samples—100% of the total possible samples in Table 1. We perform 10 cross validations on each category—90% of data is used in training the network and 10% is reserved for testing. An average MAPE of 4.3 ± 2.2 is achieved for predicting PEA estimates over all categories. The prediction error is least for energy with an average MAPE of 3.5 ± 1.2 across all categories of this subset. The average MAPE of area and performance are 4.8 ± 1.7 and 4.2 ± 2.4. EAST-DNN has 2.4× and 2× better accuracy compared to RF and XGB, respectively, which is probably attributed to the reason that DNNs are better equipped to capture the non-linear relationships between input and output data.

[1]https://github.com/tensorflow/tensorflow.

[2]RF and XGB methods are also implemented to predict PEA estimates.

Table 2: MAPE estimates for high cardinality dataset.

Image Resolution	MAPE	MAPE	MAPE	EAST-DNN accuracy over	
(W x H x C)	RF	XGB	EAST-DNN	RF	XGB
10x9x3	12.2 ± 2.6	9.8 ± 1.2	4.70 ± 2.1	2.6×	2.08×
20x9x3	10.9 ± 0.9	8.6 ± 1.4	3.77 ± 1.6	2.89×	2.28×
40x9x3	8.7 ± 2.2	7.6 ± 1.8	3.61 ± 1.3	2.40×	2.10×
40x18x3	9.5 ± 1.4	7.3 ± 0.86	3.8 ± 2.0	2.5×	1.92×
10x45x3	9.8 ± 1.4	8.9 ± 1.5	5.1 ± 2.4	1.92×	1.74×

Second, we evaluate the generalization capability of EAST-DNN by transferring a pre-trained DNN on high cardinality dataset to new low-cardinality categories with fewer training samples — 40 for each category in $\{10x18x3, 20x45x3\}$ and 95 for $\{30x9x3\}$. The pre-trained network is trained from scratch using the entire high-cardinality dataset , which yields a MAPE of 3.99 ± 1.8 for predicting PEA estimates. This model is re-trained with fewer examples from each category from the low-cardinality set—results are summarized in Table 3. In Table 3, "A" and "B" represent network re-trained with 25% and 75% of data samples from an input category belonging to low-cardinality dataset, respectively. EAST-DNN achieves 2.1× (1.7×) better accuracy, on average, against RF (XGB).

Table 3: MAPE estimates with re-trained model on low-cardinality dataset.

Image Resolution	MAPE RF	MAPE XGB	MAPE EAST-DNN	
(W X H X C)	(A)	(A)	(A)	(B)
30x9x3	15.32 ± 0.4	13.6 ± 0.78	11.6 ± 3.2	8.6 ± 3.0
20x45x3	15.84 ± 3.9	10.5 ± 0.93	7.8 ± 1.3	4.44 ± 2.1
10x18x3	23.83 ± 6.4	21.6 ± 6.0	8.3 ± 2.4	3.66 ± 1.4

Sample data collection time is nearly three weeks. It takes a minute to train EAST-DNN with complete dataset on a NVIDIA Tesla V100 GPU, followed by inference time in milliseconds. The average inference times for EAST-DNN, RF and XGB methods are 0.195 *ms*, 7.6 *ms* and 0.455 *ms* respectively. Thus EAST-DNN inference is $> 10^6\times$ faster than a single baseline simulation. In addition, EAST-DNN achieves 38.9×(2.3×) speedup over RF(XGB).

4 CONCLUSION

EAST-DNN is an efficient mechanism to massively accelerate architecture simulation with minimal accuracy loss. This is a first step towards realizing fast and accurate full system-level simulators, which can enable efficient design-space exploration of millions of design points. EAST-DNN can also be employed to rapidly explore near-optimal system designs for a target deep learning application.

REFERENCES

[1] Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A Scalable Tree Boosting System. In *Proceedings of the 22Nd ACM SIGKDD*. 785–794.

[2] Tianqi Chen and Tong He. 2015. xgboost: eXtreme Gradient Boosting. *R package version 0.4-2* (2015).

[3] Christophe Dubach, Timothy Jones, et al. 2007. Microarchitectural Design Space Exploration Using an Architecture-Centric Approach. In *MICRO*. 262–271.

[4] Jerome H. Friedman. 2000. Greedy Function Approximation: A Gradient Boosting Machine. *Annals of Statistics* 29 (2000), 1189–1232.

[5] Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *AISTATS*. 249–256.

[6] Tin Kam Ho. 1995. Random Decision Forests. In *ICDAR*. IEEE.

[7] Mark Horowitz. 2014. Computing's energy problem (and what we can do about it). In *(ISSCC)*. IEEE, 10–14.

[8] Engin İpek, Sally A. McKee, et al. 2006. Efficiently Exploring Architectural Design Spaces via Predictive Modeling. In *ASPLOS*. ACM, 195–206.

[9] D. Li, S. Yao, et al. 2016. Efficient design space exploration via statistical sampling and AdaBoost learning. In *DAC*. 1–6. https://doi.org/10.1145/2897937.2898012

[10] Yakun Sophia Shao, Brandon Reagen, et al. 2014. Aladdin: A Pre-RTL, Power-performance Accelerator Simulator Enabling Large Design Space Exploration of Customized Architectures. In *ISCA*. 97–108.