Energy Modeling of Machine Learning Algorithms on General Purpose Hardware

by

Hidayatullah Chowdary

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved September 2018 by the
Graduate Supervisory Committee:

Yu Cao, Chair
Jae-Sun Seo
Chaitali Chakrabarti

ARIZONA STATE UNIVERSITY

December 2018

ABSTRACT

Artificial Neural Network(ANN) has become a forbearer in the field of Artificial Intelligence. The innovations in ANN has led to ground breaking technological advances like self-driving vehicles,medical diagnosis,speech Processing,personal assistants and many more. These were inspired by evolution and working of our brains. Similar to how our brain evolved using a combination of epigenetics and live stimulus,ANN require training to learn patterns.The training usually requires a lot of computation and memory accesses. To realize these systems in real embedded hardware many Energy/Power/Performance issues needs to be solved. The purpose of this research is to focus on methods to study data movement requirement for generic Neural Network along with the energy associated with it and suggest some ways to improve the design.Many methods have suggested ways to optimize using mix of computation and data movement solutions without affecting task accuracy. But these methods lack a computation model to calculate the energy and depend on mere back of the envelope calculation. We realized that there is a need for a generic quantitative analysis for memory access energy which helps in better architectural exploration. We show that the present architectural tools are either incompatible or too slow and we need a better analytical method to estimate data movement energy. We also propose a simplistic yet effective approach that is robust and expandable by users to support various systems.

ACKNOWLEDGMENTS

I am grateful to my advisor Dr. Yu Cao of ECEE at Arizona State University for firstly inspiring me to pursue research and giving me this opportunity to work alongside him. I will fondly remember his lectures and the brain storming discussions with him.

I also like to thank Dr. Jae-Sun Seo and Dr. Chaitali Chakrabarti. They have been instrumental in steering my research in the right direction. I learnt basics from Dr. Seo's course and they have been vital for my research. Dr. Chaitali has always been supportive and would not hesitate to reach more researchers to find quick solutions.

My research became more fun with the help of my lab colleagues who were available to discuss my research and suggest pitfalls they have encountered. Zheng Li,my partner in my research was instrumental in understanding my research in greater detail.

I would like to thank my friends who made a positive impact in my life. Last but not least, I am thankful to my parents,my brother who have always believed in me and supported me in my pursuits.

TABLE OF CONTENTS

Page

CHAPTER Page

LIST OF TABLES

Table Page

LIST OF FIGURES

Chapter 1

INTRODUCTION

Artificial Intelligence(AI) has already become a significant part of our day to day life. These applications presently run on cloud because of the massive computation need. But many applications need real time performance and thus the trend of in-place computation is the future norm. This not only precludes to Inference tasks but also with training because a neural network which adapts to every day change can cater our needs. This thought has made researchers seek and improve techniques like reinforcement learning Mnih *et al.* (2013). On the other hand, the cloud based computation for many applications by tech giants like Google, Microsoft also require huge power and energy needs.Li *et al.* (2017) estimates that by year 2020 data centers in US alone will consume roughly 140 billion kilowatt-hours annually of which majority of applications will be that of Artificial Intelligence.

The ANN's of today are capable of many image and voice recognition tasks but are heavily relied on the computation resources of data centers. These applications cannot run seamlessly on mobile platforms because of its heavy computational requirement for a given throughput. While this is slowly changing by new architectures like Faster RCNN,Ren *et al.* (2015) and quantized architectures as presented in Howard *et al.* (2017),Wu *et al.* (2016) but even these require careful hardware planning to make an efficient system. But, these form only part of a small subset of application space.

Presently many AI applications are not realizable to run real time on mobile platforms because of hardware limitations.

1.0.1 Motivation

Energy consumption for data storage and data movements can be more than computation. This strongly depends on the data flow model, the computational throughput and the architecture of ANN used to realize the application. Canziani *et al.* (2016) summarized that state of the art neural networks require millions of parameters to be stored and trillions of operations per image. Chen *et al.* (2016) measured on hardware that typical energy of memory operation is 100 times of that of computation. This shows that computational energy is comparable with that of memory storage or memory access energy and in order to deploy AI applications on mobile devices there needs to be careful study of both computation and data movement. Many researchers have addressed these concerns with different data flow models to make efficient systems for FPGA/ASIC. Researchers also looked at better architectures for CPU, GPU to make them efficient. These works generally hypothesize a variation of implementation or a different neural architecture, implement in software and try to reach state of the art accuracy or even more with their new approach. When validating performance and efficiency of this new approach they have to either implement in FPGA or have to fabricate a custom ASIC. Prototyping the implementation to study energy/power profile would reduce save time and effort.This would also help in tweaking the design to get better results or to study memory accesses in greater detail.For computation modeling there are well known simulators both in FPGA/ASIC and CPU/GPU domain. But there are limited simulators that concentrate on data movements and the costs associated with that.Our work is to specifically understand the relationship of data accesses to and from DRAM memory and the energy/power associated with it.

1.0.2 Problem Description

To study data movements between main memory(DRAM) and rest of the system and the overall energy costs associated with it for a generic ANN training/inference.In generic sense,the system could be any of CPU,ASIC or an FPGA.Many factors affect the energy and power costs of DRAM accesses. For eg, the cache system and the parallel cores computing in case of CPU/GPU, the buffer sizes, their arrangement and the processing elements connected to it in FPGA/ASIC.As a starting point,we chose CPU based system with two levels of cache which runs popular state of the art Neural Networks. We want to quantify this energy cost, by keeping the same system and varying the ANN architecture.

1.0.3 Prior Works

To the best of our knowledge, we did not find many methods that simulate DRAM access energy for ANN applications. Many researchers use raw methods such as the total main memory accesses based on the data flow architecture and provide an estimate. Only Yang *et al.* (2016) did a formal data movement analysis and applied energy aware pruning to minimize energy consumption. They also created a website that projects the energy operation for a user given deep neural network(DNN) when it has only certain type of layers(convolution,fully connected).Their analysis does not include sequence of DRAM accesses or cache/buffer structure before the DRAM. Also, this work is function of certain inputs rather than a simulation tool which researchers can work on it and advance further.

1.0.4 Challenges

Gem5 (Binkert *et al.* (2011)) is a widely used computer architectural tool helping researchers analyze new architectures to study their performance. It provides with a suite of pre-compiled workloads which have been benchmarked on different platforms. Similarly, we picked Tensorflow (Abadi *et al.* (2016)), a software tool that is popular to run ANN applications.

Method I

Our initial method was to use tensorflow as a workload to the gem5 system and use a memory simulator to get DRAM access energy costs. Gem5 can be run in either System Emulation(SE) mode where gem5 emulates the software stack(OS) or in Full System(FS) mode you can provide an OS inside a virtual environment. Gem5 requires workload to be in executable(binary format) to run using SE mode. **?** is a utility that converts python code to executable binary. But the tensorflow binary was not able to run on gem5. We also tried FS mode,but tensorflow fail run. This is due to multiple version dependencies that tensorflow needs and the gem5 supported OS is not having those dependencies. We also noticed long simulation delays for simple programs in FS mode.

Method II

Instead of tensorflow, we wrote c based implementations for few neural networks(Lenet-5 LeCun *et al.* (2015),vgg16 Simonyan and Zisserman (2014)). The executables were passed as workload to gem5. This was successful but we observed a run time performance problem in this approach. For eg, a single layer of vgg16 took 8 hours to simulate using this setup(with sufficient cache). The cache sizes selected were same

as in high end Intel chipsets. This made us realize that we need to find a smarter solution.

1.0.5 Our approach

The computational nature of Neural Networks is unlike many other general applications. This is because we use repeated operations over and over in a sequence to realize the functionality. For eg, a 3x3x3 filter convolving with an 226x226x3 image would undergo 1354752 MAC operations spaced in between by branch operations in a single threaded,single core CPU based machine. This analogy is very true for GPU as well as FPGA.

These series of core operations when viewed as a sequence can be represented as a compressed form:

{MACS}27-{BRANCH}-{MACS}27

We can approximate the latency of core operations because of this repetitive nature of operations. Also as we are interested in memory operations, this assumption would not affect the sequence of memory operation since modern computers adhere to Vonn Neumann Architecture and would maintain memory consistency and give reproducible outcomes. Using this approximation we made a memory access model. In short, the idea is that in c based neural network codes, we skip operations instead we use observed standalone latency related to these operators. The memory operations are passed as an argument to trace_read and trace_write functions. These functions determine the virtual address of the variables involved and generate traces. Using this traces, cache models determine the final trace to the main memory. A static memory allocator is used that determines the virtual address space of each variable.This methodology is explained in detail in chapter 7.

Chapter 2

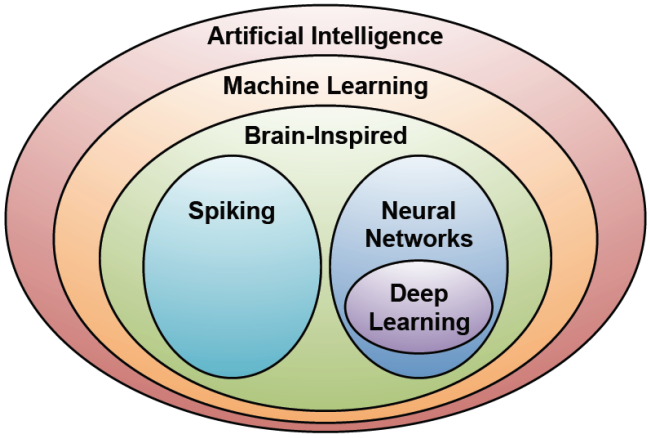
BACKGROUND

2.1 Background

Scientists have long wondered how a human or animal brain works. Many of the early studies were more towards human psychology. The first contribution to the physiology of brain came from Luigi Galvani in the second half of the 18th century. He discovered the role of electricity in dissected frog nerves. Followed by this many scientists worked on correlation between cognitive behavior of animals and their neural activity. All this fell into a discipline called Neurophysiology.

Artificial Intelligence is a field of science that explores in making intelligent machines. A subfield of this called Machine Learning deals with making machine learn on its own without being programmed. Taking inspiration from brain to master learning has been one of the popular methods. The basic element in the brain is the neuron. A human brain consists of billions on neurons. Each neuron is connected to other neurons using a structure called dendrite. With the limited knowledge on the workings of the brain,it is found that the different ionic movements form action potentials which act as processing signals. This signaling in the neuron is called firing. Since a neuron is connected to many other neurons. The behavior on this affects other neurons in two ways. When a neuron fires more because of a connected neuron firing it is called excitatory signaling and conversely if a neuron fire less because of a connected neuron firing it is called inhibitory signaling. A neuron receiving connections from other neurons sums this up and would fire more or fire less based on the temporal firing activity.

Figure 2.1: Subfields of Artificial Intelligence

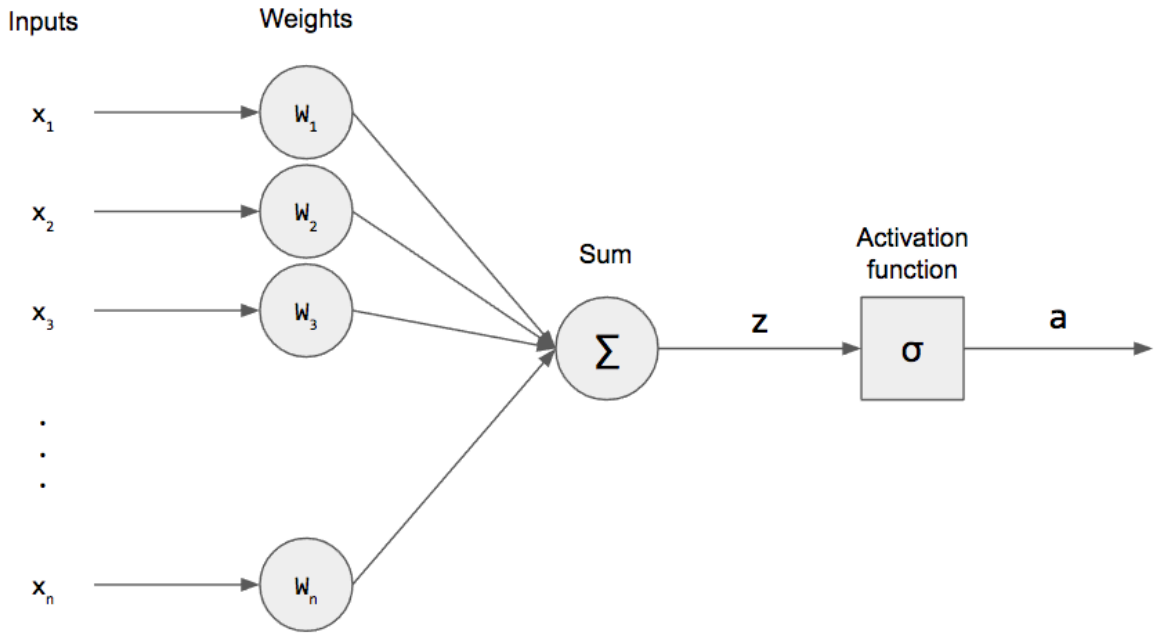


Using this primitive knowledge of workings of a brain two school of thoughts emerged for this brain inspired learning. One that closely resembles how neurons communicate called Spiking Neural Networks. Other that is a mathematical simplification of this communication called Artificial Neural Networks.

2.2 Perceptrons

The earliest neural networks models were perceptions. The excitatory and inhibitory connections are represented as connected strength(weights). Initially they were being used to store linear separable functions. The fact that this was altogether a linear function and a lack of clear strategy to train this weights by having only feed forward networks made this fail for image recognition tasks. Nonetheless this became a good starting point to explore more.

Figure 2.2: Perceptron Model

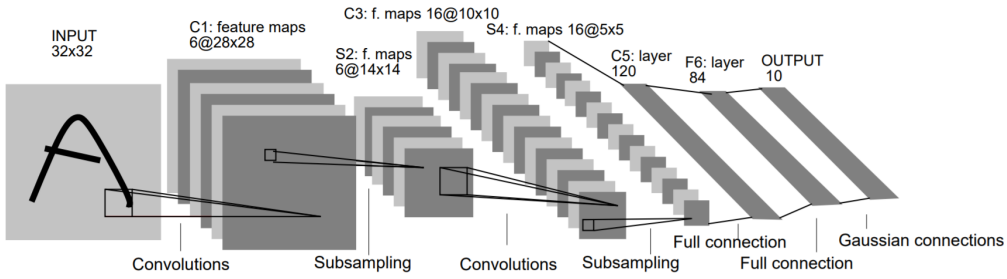


2.2.1 Break Through: Lenet-5

LeCun *et al.* (1995) was the first Neural Network model which employed back propagation. Using this, more complex networks were able to train. This popular application was used for hand written digit recognition.

His work introduced many key new concepts like convolution, sampling. It also introduced to back propagation technique. Because of lack of computational power at that times, this did not became so popular,until AlexNet won the Image Classification challenge in 2012. After that many new architectures came to improve the accuracy of image and speech recognition tasks.

Figure 2.3: LeNet 5

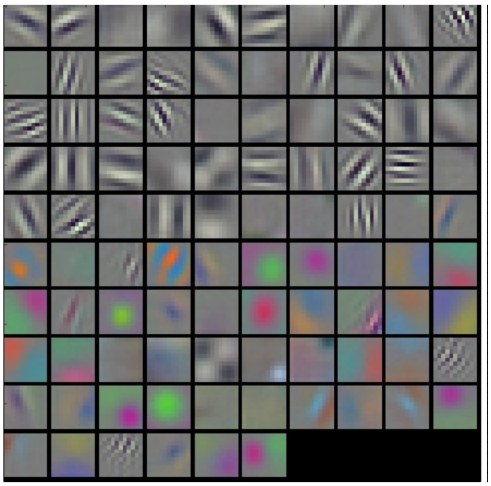


2.2.2 Brain inspiration

Torsten Wiesel and David Hubel worked on visual cortex of cat and made many important discoveries. Neural recordings of single brain cells of cats were recorded and then were able to show a topographical map in the visual cortex that represents the visual field, where nearby cells process information from nearby visual fields. The visual cortex neurons are arranged in a specific manner.Cells with similar operations are organized into columns, and these neurons relay information to a higher region of the brain, making a visual image. They found cells that respond to stimulus of specific orientation.

Convolution Neural Networks are formed from a similar principle. A set of neurons when trained would positively respond for specific edges. The correlation between the stimuli and neurons is done by convolution operator, hence the name convolution neural network. These set of neural groups are chained in a hierarchical fashion to map complex specific shapes and colors. Thus these networks can detect images based on the training dataset.

Figure 2.4: Visualization of filters of 1st conv layer of AlexNet



2.2.3 Popular Neural Network Approaches

With renewed interest of using Neural Networks for Image Classification tasks many different approaches have emerged. Some to improve the accuracy and few others to reduce the computation complexity.

While the accuracy on state of the art neural networks is beyond human performance, hardware constraints are limiting it on mobile platforms. So, researchers are focusing on new methods to cut costs on energy.

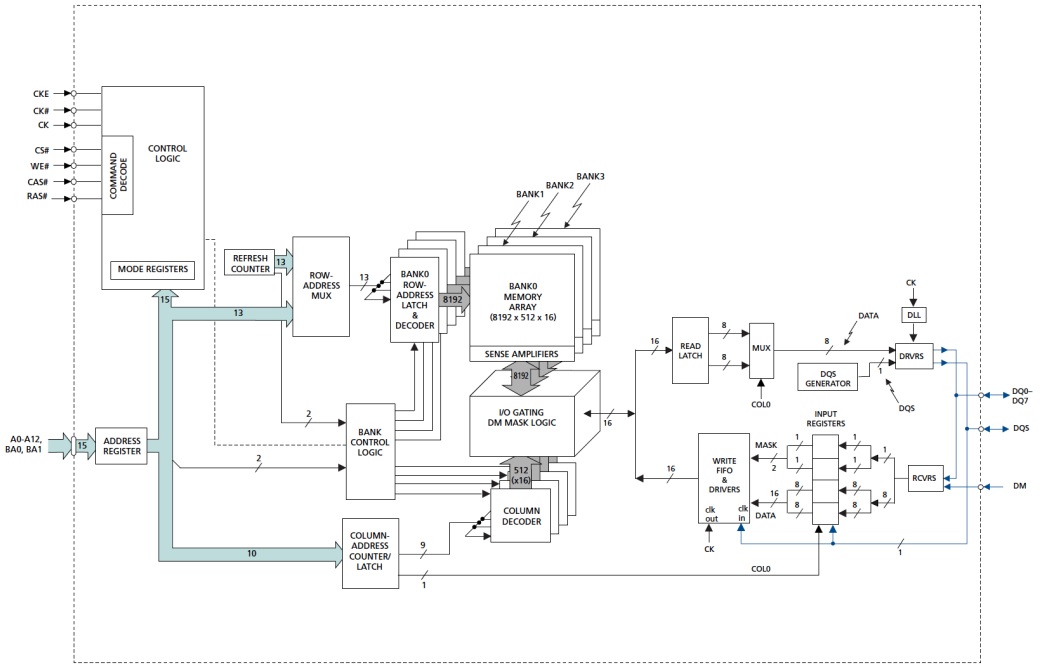
Chapter 3

MEMORY

3.1 Structure of DRAM

Before diving into the intricate details of DRAM operation we have to look into its structure thoroughly. DRAM are the defacto standard for the main memory in modern computers.

Figure 3.1: Block diagram of a typical DRAM



RAM consists of a two dimensional memory array of DRAM memory cells. DRAM memory cell uses a single transistor as a switch to charge or discharge the capacitor in series with it. Because of the simple structure it can be made dense in a compact area. The disadvantage with this is that the capacitor leaks with time and thus needs periodic refreshing.

This two dimensional array can be represented as rows and columns and termed as single Bank. Banks are arranged in a parallel manner to increase throughput by being able to access different command requests simultaneously. These banks form part of a single DRAM chip. At a time, only data from one bank is accessed onto the DRAM chip pins. The chips are grouped together to form a Rank,this will maintain data transfer width with the bus. For eg, eight DDRx8 chips form one 64 bit Rank. Likewise there will be multiple Ranks in a DIMM to increase memory capacity.

3.2 Operation of DRAM

At an abstract level, the data in the bank is accessed using a combination of row and column address. The DRAM controller is in charge of scheduling these commands and weighs it based on the state of the bank. Activate command will access the contents of the row selected and put into a row buffer. Among the row buffer cached, a column is selected and thus column access is performed. This column access is done by the RD(read) and WR(write) commands. When a row is open for accessing data, the bank is said to be in active state. Once a row is in the buffer any subsequent address access pertaining to the same row will incur low latency and energy. A row cannot be open all the time, due to the nature of DRAM as it needs periodic refreshing. The row buffer needs to be copied back to the array. At this stage the bank is said to be in Precharge state.

Based on page policies, the row can be automatically precharged after every Read or Write access, termed as RDA, WRA commands. This policy is called closed page policy. On the contrary open page policy dictates that row can be active till either an access to a different row or a periodic refresh occurs. The choice of policy selected depends on the nature of data access patterns. Open page policy helps sequential

access patterns whereas closed page policy helps random access patterns since every read or write access will observe the bank in a precharged state.

The DRAM operation can be understood in more detail by the help of these commands and states. The controller is responsible for maintaining the sequence of commands which includes the periodic refreshing. The memory controller typically will have a command queue which stores any outstanding requests which it cannot process at that particular moment.

Figure 3.2: States and commands in DRAM

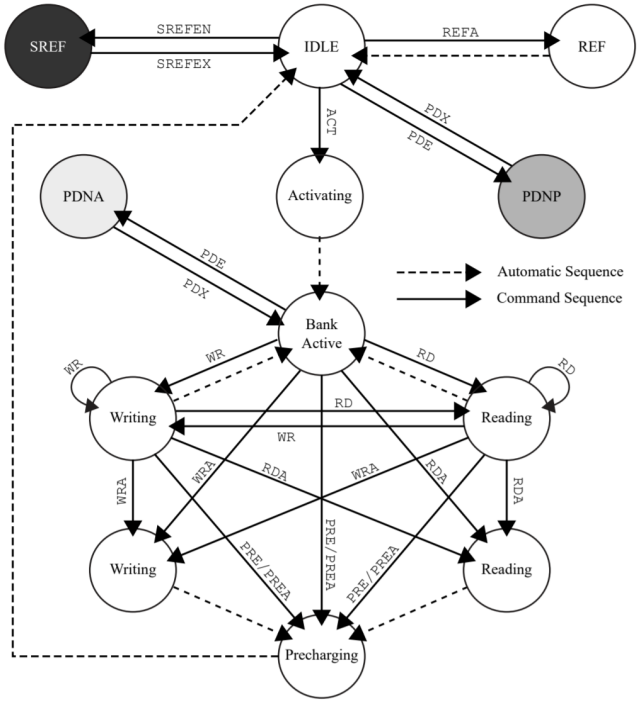


Figure 1: State diagram of DRAM commands according to JEDEC [33], power-down modes highlighted in grey [23]

Based on the above discussion we can term the different energies in a DRAM as:

- Precharge Energy
- Burst Energy
- Refresh Energy

- Background Energy

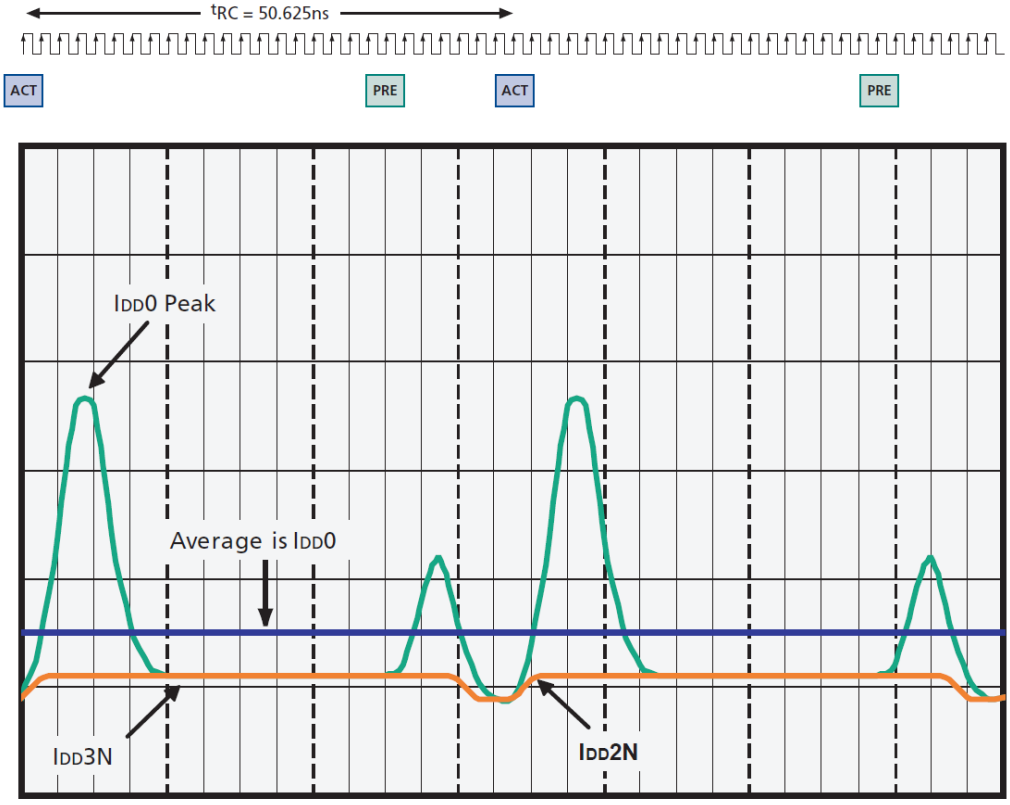
3.2.1 Currents in DRAM:

These are the current components in DRAM:

- IDD2P : banks precharged(CKE low)
- IDD3P : any bank active(CKE low)
- IDD2F banks precharged(CKE high)
- IDD3N banks active(CKE high)

IDD0 - value specified in the data sheet is the average current required for device.Based on the application, these currents change and this determines the energy dissipated. All these quantities are standardized by JEDEC (2012), which is a standardization body. The dynamics of these energy quantities is important to understand which can be controlled and which cannot be.For eg,applications which are very slow, have mostly background power which can be reduced by switching off the power to the dram. As seen from figure 2.3 that sequence of ACT and PRE commands will vary the IDD0 values which determine the power consumption of the DRAM chip.

Figure 3.3: Typical currents in DRAM



3.2.2 Bank based control

3.2.3 Background of Metrics

In the last decade many memory simulators have spawned to solve the intricacies of Memory energy and performance. Few(Nvsim2) are more concerned with emerging memory technologies like NVM memories. Other few dissected DRAM controller operations and JEDEC standardized current values to predict both power and performance.

3.2.4 Choice of memory simulators

Among these, we went with DRAMSim2 as the memory simulator. This was a follow up of DRAMSim simulator which was cycle based whereas DRAMSim2 is

event based simulator and hence improved runtime.The basis for the simulator was the TN401 DRAM Power document from Micron.

3.2.5 DRAMSim2 operation

It uses the same principle we discussed in the memory section. It designs a controller which controls the incoming data stream. Using the latency values and current values, the energy and power is calculated. The configuration for the memory is a parameter file and so is the processor parameters.

Gem5 is a discrete event driven computer system architectural simulator. This can be used to study hardware system trade-offs involving different core, cache, memory configurations with respect to different workloads.

It has a library of various pre-compiled workloads to simulate real software interactions. Architectural researchers use these to prototype a new system or subsystem and study its strengths and weaknesses.

As our aim was to simulate neural network workloads. We compiled a generic workload using tensorflow. This is not straight-forward as Gem5 requires workload as an executable. Tensorflow requires different softwares to make it work.

3.3 Executable for tensorflow:

By using an external software that takes different libraries we were able to make an executable for tensorflow.Even with this, under Full System mode of gem5, we were not able to run the executable. This is because of complex dependencies for a complex software like Tensorflow and also because gem5 full system binaries available were pretty old (2014) which caused the mismatch.

3.3.1 Advantages

Flexibility: If it had worked it would be a flexible choice for number of applcations. Accuracy: As it is cycle based and has O3 level modeling, it will have the most accurate solution SystemC support:This will enable to run generic networks on specific architectures.

3.3.2 Gem5 workload modeling

Generally gem5 is used for architecture exploration it has a number of predetermined workloads . Though many works have used it for workload modeling, there seems to be some flaws/limitations in using gem5 for workload modeling.

BLAS: BLAS3 implementation(blocking). A more realistic model.

3.3.3 PIN

We also used PIN an Intel profiling tool that gives trace of all activties of the hardware. We ran an empty tensorflow code and gave a dump of 150 GB.

3.3.4 Approach 2: C code with gem5

The other approach we sought out is to make basic c language constructs of neural networks and run the inference and training tasks in gem5 in combination with DRAMSim2.

Gem5 comes with a DRAMSim2 extension. Using this extension you can run gem5 and the memory traces from the system will be used by DRAMSim2 to generate Power/Energy.

For this Neural Network was implementation in c. The experimental results are presented in chapter 7.

Disadvantages: We observed still some drawbacks using this approach. BLAS support? Precision of the results? Efficiency in terms of simulation time Precision operations - how much support? FPGA based support?

3.3.5 Need for Memory Access Model

Sequence of memory accesses are important than just the overall reads and writes. To demonstrate this we did a small experiment where in we accessed a contiguous part of memory in two different ways.(i) sequential access,(ii) random access.For 10,000 iterations here is the outcome in table , that sequential accesses consume less power than non sequential access. This is because in the open page policy, bank row will be open and both latency and energy of access is better for accesses to the same row.

Chapter 4

METHODOLOGY

4.1 Methodology

In order to model memory accesses for an ANN we came with this configuration and computing approach as shown in fig 6.1. Using a network graph either using a prototxt file or a configuration code, the network topology is recorded. The Compute and Memory access Model traverses through this topology and generates memory traces along with rough timing information. The read or write trace is processed by a two level cache model.From the cache model we get DRAM memory traces which are fed to a memory simulator.In a general purpose processor we are aware that cache hit/miss rate is crucial for performance and efficiency of the overall system. This cannot be more true for a neural network application.

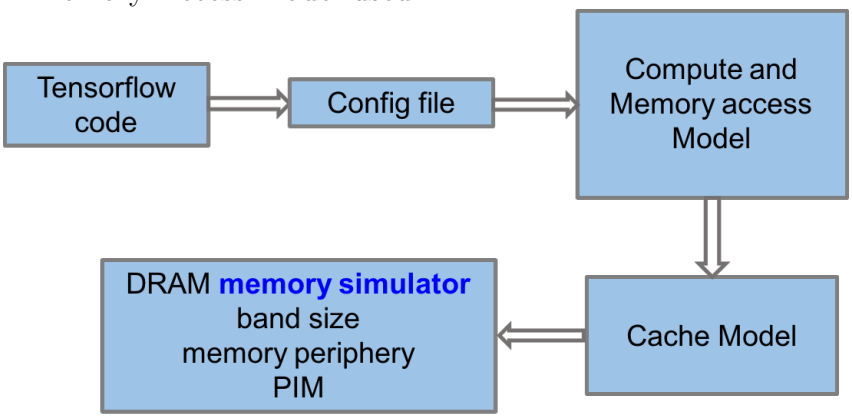
As we are focused on generic Neural Network Training we made c based layer models which support general neural Net layers like: Convolution,ReLU,Pooling, FC,dropout, softmax, batchnorm, concat. The reads and writes to the address locations and the timing of these transactions affect the memory access energy. The sequence of read/write access is a fine control to the energy outcome. This is based on the page policy used. And as discussed in section 3.3 the general processing uses open page policy, which makes sequential memory access both efficient and fast.

4.2 Overview

The modeling software is divided into mainly four blocks, we will go through each of it in more detail.

- Application Interface
- Compute Model
- Memory Model
- Memory Simulator

Figure 4.1: Memory Access Model used



4.3 Application Interface

To be able to make this analysis general to most of neural networks, there has to be a way to communicate the network topology. This can be done either using a prototxt file used in caffe or network file as per tensorflow. Presently, we used a c based node configuration format where each c string parameter represents the inputs,parameters,outputs through a particular layer denoted by the c function name.The non-string parameters represent the input map size, filter map size, number of filers.

conv_3d(fp,fp2,"in","filter1","fmap1",226,3,3,64,1);

Simiarly all layers have similar way of representation.

4.4 Compute Model

Sequences of reads and writes to memory locations is important for memory energy calculation. The timing information is important too as it decides the standby

energy as well as the power dissipation of the RAM. This scheduling is done differently in different tools.We chose an open and simple approach for this as it allows flexibility and robustness to the simulation semantics. The general computation latency from the manufacturers can be configured into a file and this can be used for latency calculation. This approach provides flexibility to simulate systems from different vendors.The downside to this or any architecture simulator is that the micro architecture dictates the percent of mismatch between the real and simulated results. Gem5 is a popular architectural simulator which predicts performance close to real systems using its out of order O3 model. Generally these simulations take 10,000 times longer run time than the real simulation. If we take a reasonable neural network which runs few seconds in a multicore machine but we need to consider it interms of a single core as the simulator can run on a single core. So a 10 second application would take 100000 seconds, that is approximately 27 hours.

Our argument is based on the fact that ANNs do majorly long repetative computations either it is a MAC operation,comparator operation and we can leverage this to simplify the model to run faster. Infact, by knowing microarchitecture details,precise memory access timings can be derived by a small change in code.

The latency of an operation is stored in variables using the projected charts.The primary computation in neural network is a Multiply and Accumulate(MAC) operation. Each vendor has its own optimized solutions for these type of operations. Like, Intel's AVX, AVX 512 use heavy vector operator SIMD operations to speed up. This can be modeled by changing the loop structure inside layer model code without any architectural change.

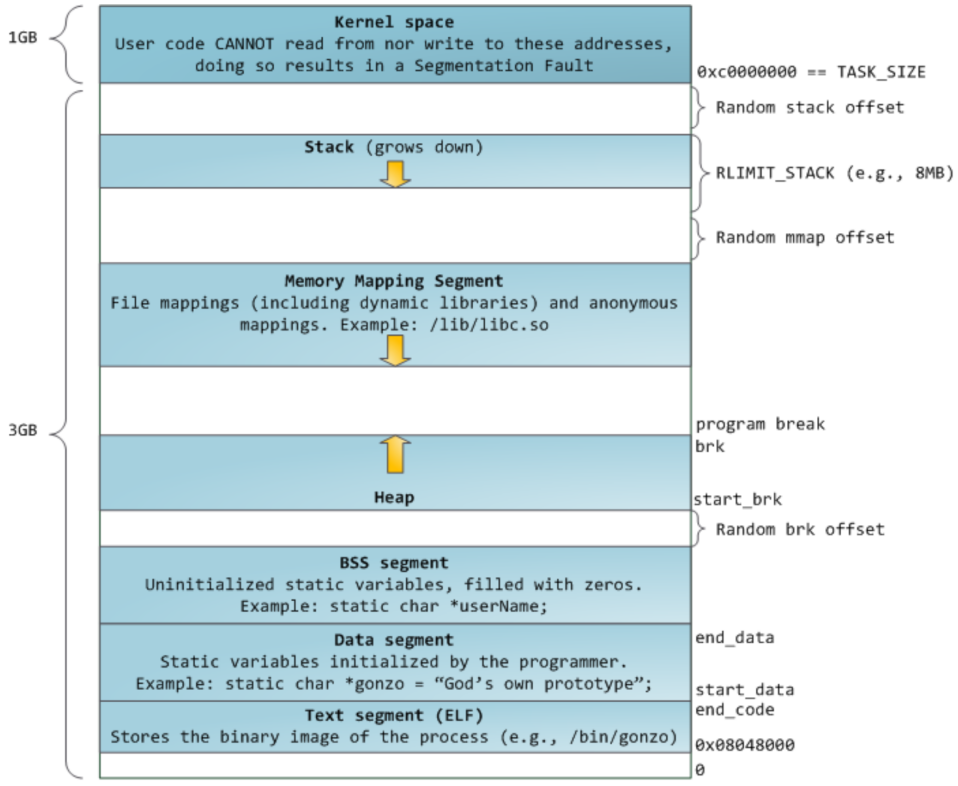
The compute Model is similar in comparison with Timing Simple model of gem5 which follows execute-in-execute order which takes care of any dependency and thus maintains simulation accuracy.

4.5 Memory Model

4.5.1 Memory Mapping

It is important to understand how variables used in a program map to the virtual memory(general PCs) and to physical memory(embedded systems). In general-purpose CPUs, the program resources are mapped into a virtual address space. This is not the case presently in small embedded systems where resources are mapped directly to the physical memory. The newer embedded systems are willing to support virtual memory to support more applications.

Figure 4.2: Virtual Address Space



The virtual address space is divided into region as shown by the figure 6.1.Instead of dealing with an executable to leverage the storage elements required we directly

use the c code and the language constructs to map memory elements to particular addresses with sufficient memory space.

For example,to represent activation maps and weights, generally we need to use malloc construct to dynamically allocate memory which goes into the heap region. Similarly the temporary variables in the code is placed in the stack part. This distinctive regions are maintained to avoid collisions between different variable memories and thus prevent segmentation faults.

For simplicity, we map the virtual address space directly to the RAM address space. This assumption works when your working address space is within the RAM size. Moreover, in actual systems the mapping of virtual to physical address is done using combination of software(OS kernal) and hardware(TLB,page table). We do not think, either presence or absence of this feature will impact the memory access energy as this allocation depends on history of other processes run on the system and thus random in nature. This overlooks the limitation of frame size, which when included would make memory chunks discreetly continuous and might give slight deviation in the outcome.

4.5.2 Cache Model

Cache models with variable configurability like cache block size,cache size,write back/write through policy is used. This also has line based prefetcher.For our simulations we have only used tow levels of cache. This can be extended by instantiation.

Chapter 5

RESULT AND OBSERVATIONS

5.1 Factors affecting Memory Access Energy

At the start of the AI revolution researchers were more focused on tapping its potential i.e Image Classification Accuracy or Object Recognition Accuracy. At the same time hardware focused research happened to accelerate the inference and training tasks. At that time much thought was not given to making energy efficient systems.To encourage researchers for efficient

5.2 Access Pattern Matters

This experiment was done to show that the pattern of memory access is important in determining the power consumed.We wrote a code which does 10,000 reads in a range of memory.

Figure 5.1: Sequence vs Random Pattern Accesses

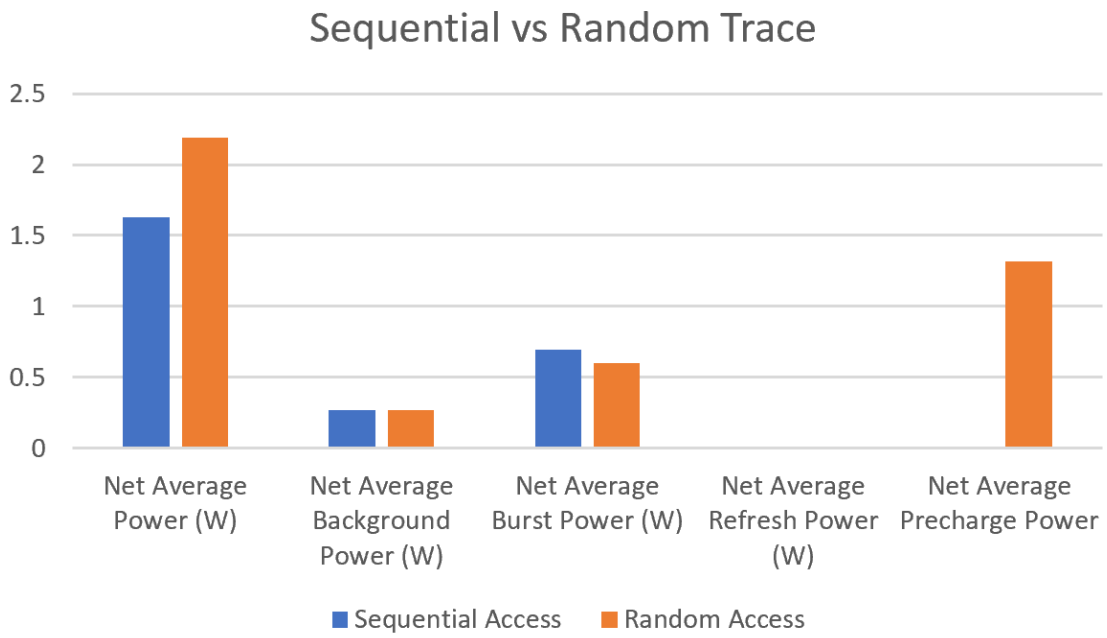


Table 5.1: Sequence vs Random Access DRAM Energy

component	Metric:Energy(mJ)	Metric:Power(W)
Average	1.62	2.19
Background	0.26	0.27
Burst	0.69	0.59
Refresh	0.009	0.009
Precharge	0.009	1.32

In the first case we read the memory locations in sequence. In the second case we read the memory locations within the range in a random fashion.

This shows that the precharge power is magnitudes more than in case of Random access compared to Sequential Access. We can also observe that the burst power is

more for sequence access as the latency is less for sequence accesses. If we normalize across the duration of both experiments this quantity would be same.

5.3 Exploring cache configuration based on application

In this we used this tool to explore different system configurations to determine an optimal solution. Based on the energy budget one can choose which cache to use in their system.

5.3.1 L1:16KB,L2:128KB

Table 5.2: Vgg16 Inference Power-Energy Table(L1:16KB,L2:128KB)

component	**Metric:Energy(mJ)**	**Metric:Power(W)**
Energy	3832.10	1.344
Background	690.86	0.269
Burst	1472	0.57
Refresh	24.98	0.009
Precharge	1643	0.64

Table 5.3: Vgg16 Inference Power-Energy Table(L1:32KB,L2:256KB)

component	**Metric:Energy(mJ)**	**Metric:Power(W)**
Energy	3296.37	1.344
Background	658.57	0.268
Burst	1152	0.47
Refresh	23.92	0.009
Precharge	1461.228	0.59

Table 5.4: Vgg16 Inference Power-Energy Table(L1:64KB,L2:512KB)

component	**Metric:Energy(mJ)**	**Metric:Power(W)**
Energy	2936.72	1.22
Background	637.78	0.26
Burst	993.35	0.414
Refresh	23.92	0.009
Precharge	1282.19	0.534

Table 5.5: Vgg16 Inference Power-Energy Table(L1:64KB,L2:1MB)

component	**Metric:Energy(mJ)**	**Metric:Power(W)**
Energy	2075.83	1.39
Background	396.33	0.266
Burst	800	0.53
Refresh	14.51	0.009
Precharge	864.9	0.58

5.3.2 L1:32KB,L2:256KB

5.3.3 L1:64KB,L2:512KB

5.3.4 L1:64KB,L2:1MB

Figure 5.2: Energy for different cache configurations

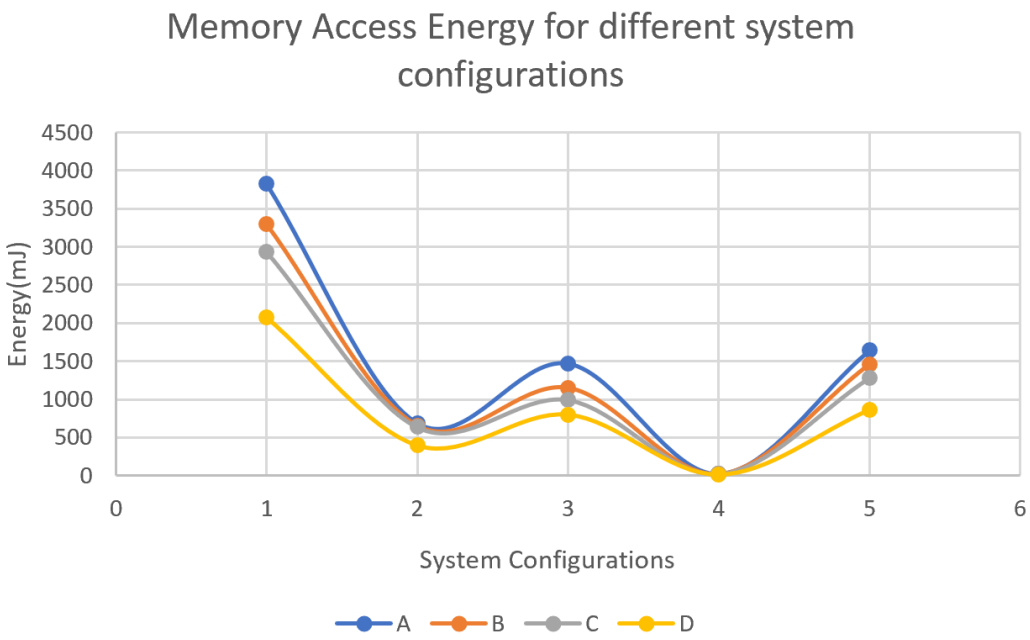
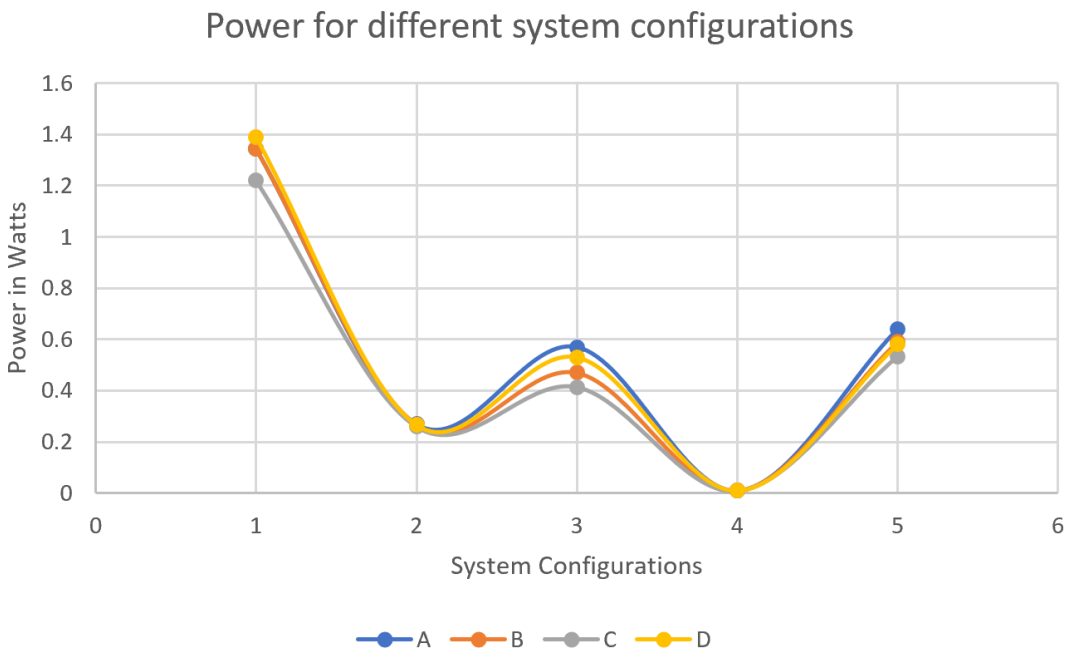


Figure 5.3: Power for different cache configurations



5.4 Runtime of Simulations

We observed based on our Approach 2(c based neural network with gem5+DRAMSim2), the simulation speed were drastically slow. For a 6 hour duration we are able to run single layer of Vgg-16 inference. This below table shows a relative comparison of the simulation runtime for small work loads.

Table 5.6: Runtime comparison between gem5 vs our script

Operation	**gem5+DRAMSim2(mins)**	**model+DRAMSim2(mins)**
10x10x3 conv 3x3x3x64	1.9	1
50x50x3 conv 3x3x3x64	84.15	2
50x50x3 conv 3x3x3x64	146	4

5.5 Results from Literature

Other research related to energy in ANN was not as fine tuned as this but was at a broader level wherein they report the overall energy usage. For example, the energy efficiency by Nvidia's whitepaper have shown that AlexNet ran with a batch size of 1 with corei7 6700K(FP32) gave a value of 1.3 img/sec/W. Our simulation results fall in the range of this real energy efficiency. Although this is not exactly one to one comparison, that should be part of future work and would add greater value to this work.

REFERENCES

Abadi, M., P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "Tensorflow: a system for large-scale machine learning.", in "OSDI", vol. 16, pp. 265–283 (2016).

Binkert, N., B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti *et al.*, "The gem5 simulator", ACM SIGARCH Computer Architecture News **39**, 2, 1–7 (2011).

Canziani, A., A. Paszke and E. Culurciello, "An analysis of deep neural network models for practical applications", arXiv preprint arXiv:1605.07678 (2016).

Chen, Y.-H., J. Emer and V. Sze, "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks", in "ACM SIGARCH Computer Architecture News", vol. 44, pp. 367–379 (IEEE Press, 2016).

Howard, A. G., M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications", arXiv preprint arXiv:1704.04861 (2017).

JEDEC, "Jesd79-3f,ddr3 sdram standard", (2012).

LeCun, Y., L. Jackel, L. Bottou, A. Brunot, C. Cortes, J. Denker, H. Drucker, I. Guyon, U. Muller, E. Sackinger *et al.*, "Comparison of learning algorithms for handwritten digit recognition", in "International conference on artificial neural networks", vol. 60, pp. 53–60 (Perth, Australia, 1995).

LeCun, Y. *et al.*, "Lenet-5, convolutional neural networks", URL: http://yann. lecun. com/exdb/lenet p. 20 (2015).

Li, Y., Y. Wen, K. Guan and D. Tao, "Transforming cooling optimization for green data center via deep reinforcement learning", arXiv preprint arXiv:1709.05077 (2017).

Mnih, V., K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra and M. Riedmiller, "Playing atari with deep reinforcement learning", arXiv preprint arXiv:1312.5602 (2013).

Ren, S., K. He, R. Girshick and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks", in "Advances in neural information processing systems", pp. 91–99 (2015).

Simonyan, K. and A. Zisserman, "Very deep convolutional networks for large-scale image recognition", arXiv preprint arXiv:1409.1556 (2014).

Wu, J., C. Leng, Y. Wang, Q. Hu and J. Cheng, "Quantized convolutional neural networks for mobile devices", in "Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition", pp. 4820–4828 (2016).

Yang, T.-J., Y.-H. Chen and V. Sze, "Designing energy-efficient convolutional neural networks using energy-aware pruning", arXiv preprint arXiv:1611.05128 (2016).