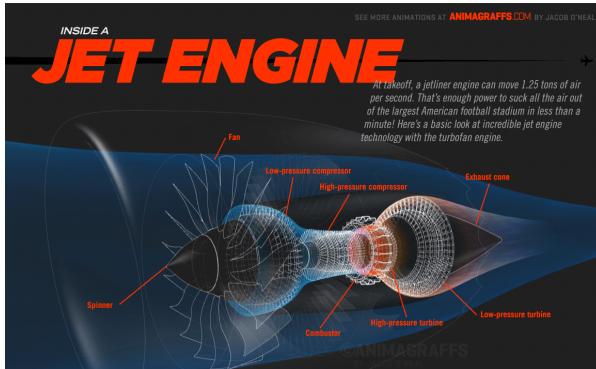


Network Compression and Acceleration

It is about how to accelerate the training and inference of deep learning(generally machine learning) except numerical optimization methods including the following topics:

- compiler optimization for computation intensive programs;
- system architecture design for computation intensive programs;
- network model compression.



- Network Compression and Acceleration
 - Resource on ML Sys
 - Workshop and Conference
 - Patents and Products
 - Courses and Labs
 - System for Deep Learning
 - Parallel Architectures and Special Hardware
 - GPU
 - TPU
 - NPU
 - Compilers for Deep Learning
 - TVM and Versatile Tensor Accelerator (VTA)
 - nGraph
 - XLA
 - JAX: Autograd and XLA
 - Multi-Level Intermediate Representation
 - Glow
 - Cilk
 - Triton
 - DLVM
 - CHET
 - NeDLR
 - Butterflies: A Universal Building Block for Structured Linear Maps
 - halide
 - taichi
 - TASO
 - Numerical algorithms for high-performance computational science

- Fast Matrix-vector Multiplication
 - Computation of Matrix Chain Products
 - Generalized Matrix to Matrix Multiplication
 - Strassen Algorithms
 - Coppersmith–Winograd Algorithms
 - Linear Algebra Packages
 - Automatic Differentiation, Differentiable Programming and Program Transformations
 - Automatic Differentiation
 - Differentiable Programming
 - Program Transformations
 - Fixed-point Arithmetic and Approximate Computing
 - High-Performance Deep Learning
- Compression and Acceleration of Convolutional Neural Networks
 - Parameter Pruning and Sharing
 - Quantization and Fixed-point Arithmetic
 - Low Bit Neural Network
 - Deep Model Compression
 - 8-bit-training
 - Binarized Neural Network, Ternary Weight Networks, XOR-Net
 - Binarized Neural Network
 - Ternary Weight Networks
 - XOR-Net
 - Mixed Precision Trainging
 - Blended Coarse Gradient Descent
 - Low-precision Training
 - High-accuracy Low Precision
 - Ultra-Low Precision Training
 - ADMM
 - Huffman Encoding
 - Knowledge Distillation
 - Transferred/Compact Convolutional Filters
 - Tensor Methods
 - Privacy and Security
 - Distributed Training
 - Accelerating Deep Learning Workloads
 - PipeDream
 - Count-Sketches
 - Efficient Communication
 - DeepSpeed
 - Gradient Code and Compression
 - Gradient Code and Approximate Gradient Coding
 - Gradient Compression

- Deep Gradient Compression @ MIT
- Gradient Compression @ epfl
- Gradient Compression @ Edinburgh
- Gradient Compression @ kaust
- Synthetic gradient
- Gradient Centralization
- Compressing Recurrent Neural Network
- Compressing GANs
- Products and Packages
- Edge Computation
 - Mobile Deep Learning
- Tool kits

The World of Neural Network Acceleration

Choice of Algorithm

Parallelism

Distributed Computing

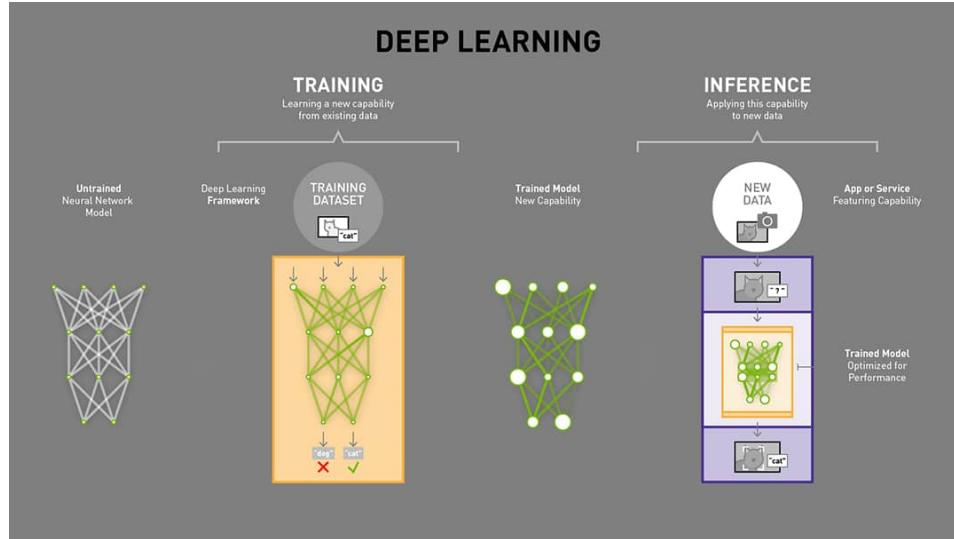
Hardware Architectures

- Trax — your path to advanced deep learning
- THE 5TH ANNUAL SCALEDML CONFERENCE
- <https://www.atlaswang.com/>
- <https://faculty.ucmerced.edu/mcarreira-perpinan/research/MCCO.html>
- <https://github.com/1duo/awesome-ai-infrastructures>
- <https://duvenaud.github.io/learning-to-search/>
- <http://www.eecs.harvard.edu/htk/publications/>
- <http://www.eecs.harvard.edu/htk/courses/>
- FairNAS: Rethinking Evaluation Fairness of Weight Sharing Neural Architecture Search
- VISUAL COMPUTING SYSTEMS
- Tutorial on Hardware Accelerators for Deep Neural Networks
- Survey and Benchmarking of Machine Learning Accelerators
- <https://zhuanlan.zhihu.com/jackwish>
- <https://machinethink.net/blog/compressing-deep-neural-nets/>
- Rethinking Deep Learning: Architectures and Algorithms
- <https://girisharma.in/teaching/efficient-cnns/>
- <https://arxiv.org/abs/1904.00938>
- <https://github.com/ChanChiChoi/awesome-model-compression>
- <https://github.com/fengbintu/Neural-Networks-on-Silicon>
- <https://vast.cs.ucla.edu/>
- Blade Benchmark Suite(BBS)简介
- <https://web.northeastern.edu/yanzhiwang/research/>
- <http://shivaram.org/#teaching>

To revolutionize deep learning with real-time AI solutions that scale from the edge to the data center.

The parameters of deep neural networks are tremendous. And deep learning is matrix-computation intensive. Specific hardware such as GPU or TPU is used to speed up the computation of deep learning in training or inference. The optimization methods are used to train the deep neural network. To boost the training of deep learning, we would like to design faster optimization methods such as ADAM and delicate architectures of neural network such as ResNet. After training, the parameters of the deep neural network are fixed and used for inference, we would do much matrix multiplication via the saved fixed parameters of deep neural network.

From [What's the Difference Between Deep Learning Training and Inference?](#)



Training

Acceleration

<https://web.stanford.edu/~perdavan/DNNTrain/>

Tutorial on Hardware Accelerators for Deep Neural Networks

Inference

Compression

<https://www.intel.ai/accelerating-tensorflow-inference-with-intel-deep-learning-boost-on-2nd-gen-intel-xeon-scalable-processors/>

Accelerating Large Scale Deep Learning Inference through DeepCPU at Microsoft

Evolution of Model Compression and Acceleration

Computer Architecture: TPUs, GPUs

Compilers: [TVM](#)

Model Re-design: [EfficientNet](#)

Re-parameterization: Pruning

Transfer Learning

When the computation resource is limited such as embedded or mobile system, can we deploy deep learning models? Definitely yes.

- Awesome model compression and acceleration
- Acceleration and Model Compression by Handong
- Accelerating Deep Learning Inference via Freezing
- 模型压缩之deep compression

- 论文笔记《A Survey of Model Compression and Acceleration for Deep Neural Networks》
- <https://zhuanlan.zhihu.com/p/67508423>
- Network Speed and Compression
- Model Compression and Acceleration
- Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding
- Deep Gradient Compression: Reducing the Communication Bandwidth for Distributed Training
- AutoML 的十大开源库
- TensorFlow模型压缩和Inference加速
- <https://zhuanlan.zhihu.com/DNN-on-Chip>
- <https://github.com/BlueWinters/research>
- Reference workloads for modern deep learning methods
- How to Train for and Run Machine Learning Models on Edge Devices
- <http://mvapich.cse.ohio-state.edu/>
- <http://people.eecs.berkeley.edu/~reddy/>

Resource on ML Sys

- <https://www.microsoft.com/en-us/research/project/astra-custom-wired-dnns/>

Workshop and Conference

- <http://www.sysml.cc/>
- <https://mlperf.org/>
- <https://sosp19.rcs.uwaterloo.ca/program.html>
- <http://learningsys.org/sosp19/schedule.html>
- <https://cs.stanford.edu/people/zhihao/>
- Workshop on AI Systems
- Systems for ML
- Workshop on ML for Systems at NeurIPS 2019
- <http://learningsys.org/nips18/>
- <http://learningsys.org/sosp17/>
- <https://sites.google.com/site/mlsys2016/>
- Workshop on Systems for ML and Open Source Software at NeurIPS 2018
- Computer Systems Colloquium (EE380) Schedule
- ML Benchmarking Tutorial
- The ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)
- The fastest path to machine learning integration
- <https://www.sigarch.org/call-participation/ml-benchmarking-tutorial/>
- DNNBuilder: an Automated Tool for Building High-Performance DNN Hardware Accelerators for FPGAs
- ISCA 2016 in Seoul
- Acceleration of Deep Learning for Cloud and Edge Computing@UCLA
- Hot Chips: A Symposium on High Performance Chips
- <https://paco-cpu.github.io/paco-cpu/>
- Programmable Inference Accelerator

- Fair and useful benchmarks for measuring training and inference performance of ML hardware, software, and services.
- <https://dlonsc19.github.io/>

Patents and Products

- <https://www.nextplatform.com/>
- <https://iq.opengenus.org/neural-processing-unit-npu/>
- <https://www.csail.mit.edu/event/domain-specific-accelerators>
- <https://www.alphaics.ai/>
- <http://www.cambricon.com/>
- <https://www.sigarch.org/>
- <https://www.xilinx.com/>
- <https://wavecomp.ai/>
- <https://www.graphcore.ai/>
- https://www.wikiwand.com/en/Hardware_acceleration
- <https://en.wikichip.org/wiki/WikiChip>
- <https://patents.google.com/patent/US8655815B2/en>
- <https://www.intel.ai/blog/>
- <https://www.arm.com/solutions/artificial-intelligence>
- <https://mlperf.org/index.html#companies>
- Accelerating Deep Learning with Memcomputing
- <http://maxpumperla.com/>
- <http://accelergy.mit.edu/>

Courses and Labs

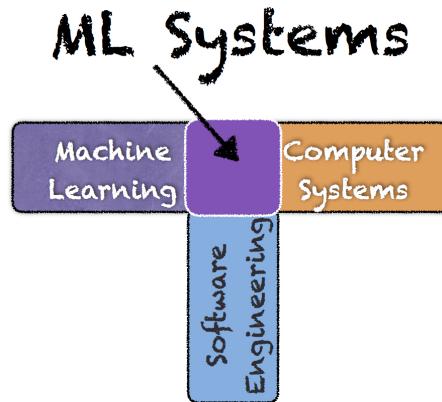
- Modern Numerical Computing
- Papers Reading List of *Embedded Neural Network*
- Deep Compression and EIE
- Programmable Hardware Accelerators (Winter 2019)
- Hardware Accelerators for Training Deep Neural Networks
- Illinois Microarchitecture Project using Algorithms and Compiler Technology
- Deep Learning for Computer Architects
- System for Machine Learning @.washington.edu/
- Hanlab: ACCELERATED DEEP LEARNING COMPUTING Hardware, AI and Neural-nets
- Bingsheng He's publication on GPU
- Architecture Lab for Creative High-performance Energy-efficient Machines
- HIGH PERFORMANCE POWER EFFICIENT NEURAL NETWORK IMPLEMENTATIONS ON EMBEDDED DEVICES
- <https://eiclab.net/>
- <https://parsa.epfl.ch/~falsafi/>
- swDNN: A Library for Accelerating Deep Learning Applications on Sunway TaihuLight Supercomputer
- High Performance Distributed Computing (HPDC)
- <https://readingxtra.github.io/>
- <https://nextcenter.org/>
- <http://yanjoy.win/>
- <https://dlonsc19.github.io/>

- <http://hibd.cse.ohio-state.edu/>
- <http://www.federated-ml.org/>
- <https://dai.lids.mit.edu/research/publications/>
- <https://aiforgood.itu.int/>
- <https://www.linayao.com/publications/>
- <https://www.deepstack.ai/>
- <https://pvs.ifi.uni-heidelberg.de/home>

System for Deep Learning

Over the past few years, deep learning has become an important technique to successfully solve problems in many different fields, such as vision, NLP, robotics. An important ingredient that is driving this success is the development of deep learning systems that efficiently support the task of learning and inference of complicated models using many devices and possibly using distributed resources. The study of how to build and optimize these deep learning systems is now an active area of research and commercialization.

Matrix computation dense application like deep neural network would take the advantages of specific architecture design. Thus it is really close to high performance computational science when solving some computation dense problems.



seq2seq	3 2 35 0 0 0 32 0 0 2 0 0 0 0 0 0 3 20 0 1 0 0
memnet	2 1 33 1 0 4 12 2 0 9 0 0 0 0 5 0 9 13 0 0 1 1
speech	0 0 0 0 0 0 89 0 0 0 0 0 0 0 0 7 3 0 0 0 0
autoenc	3 0 6 0 5 0 58 0 0 2 0 0 0 5 8 0 0 9 0 0 0 0
residual	0 0 0 0 0 0 0 0 0 33 34 32 0 0 0 0 0 0 0 0 0 0
vgg	0 0 0 0 0 0 0 0 0 35 31 30 0 2 0 0 0 0 0 0 0 0
alexnet	0 0 0 0 0 0 7 0 3 0 31 26 31 0 0 0 0 0 0 0 0 0
deepq	0 0 0 0 0 0 11 0 0 0 33 27 20 0 0 7 0 0 0 0 0 0
	Add Sub Mul Div Pow Softmax MatMul CrossEntropy MaxPoolGrad Sum Conv2D Conv2DBackFilter Conv2DBackInput RandomNormal ApplyAdam ApplyRMSPProp Transpose Tile Select Pad Reshape Shape
A	
B	
C	
D	
E	
F	
G	

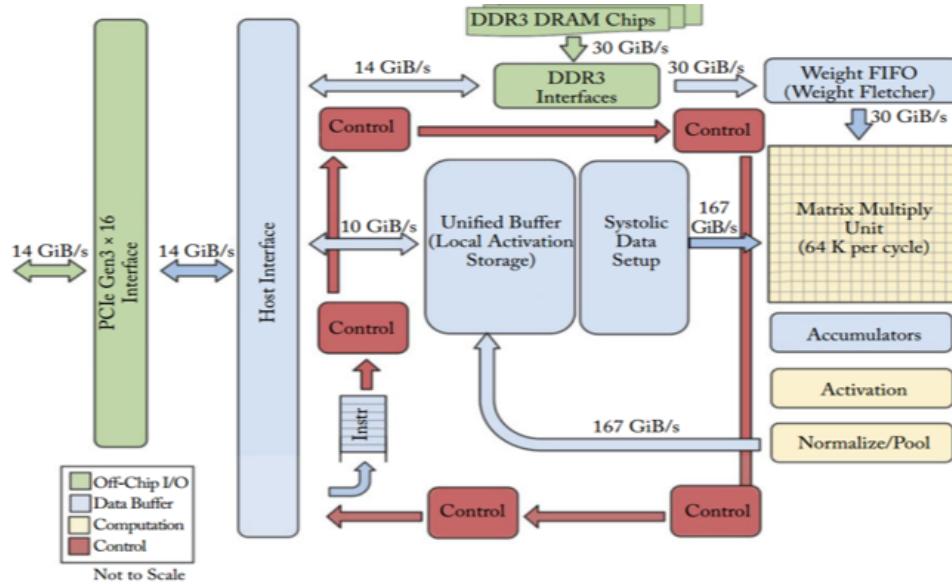
Group	Op Class
A	Elementwise Arithmetic
B	Matrix Operations
C	Reduction and Expansion
D	Convolution
E	Random Sampling
F	Optimization
G	Data Movement

- BENCHMARKING DEEP LEARNING SYSTEMS
- Facebook AI Performance Evaluation Platform
- Hardware Accelerators for Machine Learning (CS 217) Stanford University, Fall 2018
- CSCE 790/590: Machine Learning Systems
- DeepDream: Accelerating Deep Learning With Hardware
- <http://ece-research.unm.edu/jimp/codesign/>
- <http://learningsys.org/sosp19/>
- <https://determined.ai/>

- <https://web.stanford.edu/~rezab/>

Parallel Architectures and Special Hardware

Parallel Architectures for Parallel Processing as co-design is a subfield of system for machine learning.

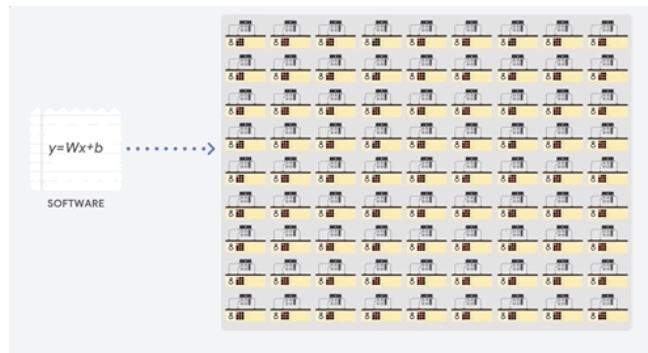


- https://vistalab-technion.github.io/cs236605/lectures/lecture_8/
- https://vistalab-technion.github.io/cs236605/lectures/lecture_9/
- https://vistalab-technion.github.io/cs236605/lectures/lecture_10/
- <https://iq.opengenus.org/gpu-vs-tpu-vs-fpga/>
- Introduction to Parallel Computing Author: Blaise Barney, Lawrence Livermore National Laboratory
- Parallel Architectures for Artificial Neural Networks: Paradigms and Implementations N. Sundararajan, P. Saratchandran
- Parallel Computer Architecture and Programming (CMU 15-418/618)
- <https://twimlcon.com/>
- Accelerating Deep Learning with a Parallel Mechanism Using CPU + MIC
- Papers on Big Data Meets New Hardware

GPU

This GPU architecture works well on applications with massive parallelism, such as matrix multiplication in a neural network. Actually, you would see order of magnitude higher throughput than CPU on typical training workload for deep learning. This is why the GPU is the most popular processor architecture used in deep learning at time of writing.

But, the GPU is still a general purpose processor that has to support millions of different applications and software. This leads back to our fundamental problem, the von Neumann bottleneck. For every single calculation in the thousands of ALUs, GPU need to access registers or shared memory to read and store the intermediate calculation results. Because the GPU performs more parallel calculations on its thousands of ALUs, it also spends proportionally more energy accessing memory and also increases footprint of GPU for complex wiring.

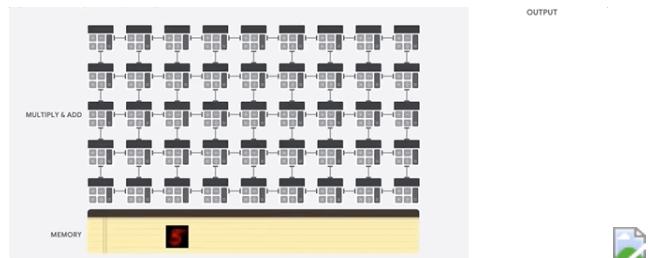


- GPU, CUDA, cuDNN的理解
- <https://developer.nvidia.com/cuda-zone>
- <https://arxiv.org/pdf/1410.0759.pdf>

TPU

TPUs can't run word processors, control rocket engines, or execute bank transactions, but they can handle the massive multiplications and additions for neural networks, at blazingly fast speeds while consuming much less power and inside a smaller physical footprint.

The key enabler is a major reduction of the von Neumann bottleneck. Because the primary task for this processor is matrix processing, hardware designer of the TPU knew every calculation step to perform that operation. So they were able to place thousands of multipliers and adders and connect them to each other directly to form a large physical matrix of those operators. This is called systolic array architecture.



- In-Datacenter Performance Analysis of a Tensor Processing Unit
- <https://www.mlq.ai/tpu-machine-learning/>
- EfficientNet-EdgeTPU: Creating Accelerator-Optimized Neural Networks with AutoML
- An in-depth look at Google's first Tensor Processing Unit (TPU)
- A Domain-Specific Architecture for Deep Neural Networks

NPU

A neural processing unit (NPU) is a microprocessor that specializes in the acceleration of machine learning algorithms, typically by operating on predictive models such as artificial neural networks (ANNs) or random forests (RFs). It is, also, known as neural processor.

NPU are required for the following purpose:

1. Accelerate the computation of Machine Learning tasks by several folds (nearly 10K times) as compared to GPUs
2. Consume low power and improve resource utilization for Machine Learning tasks as compared to GPUs and CPUs

Compilers for Deep Learning

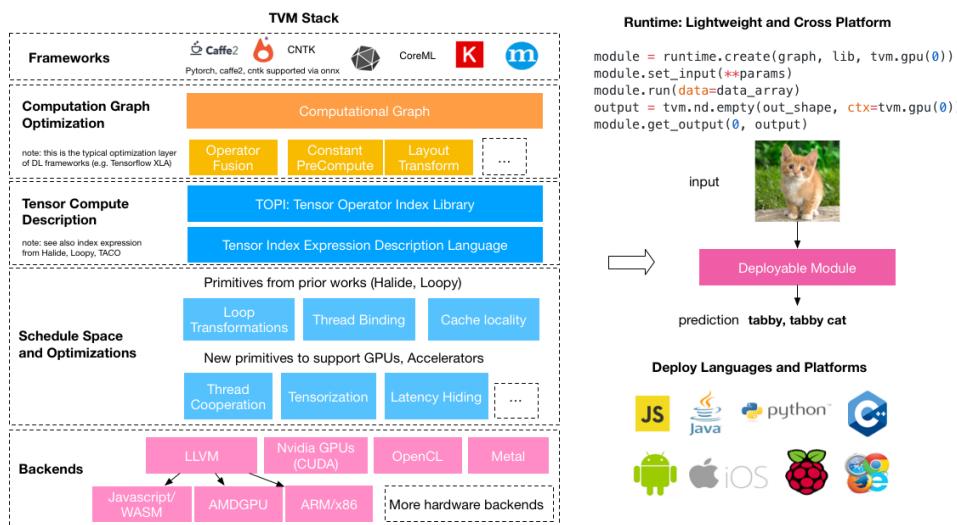
- <https://aiotworkshop.github.io/2020/program.html>

- <https://www.incose.org/>
- Compilers by SOE-YCSCS1 STANFORD SCHOOL OF ENGINEERING
- EIE: Efficient Inference Engine on Compressed Deep Neural Network
- A modern compiler infrastructure for deep learning systems with adjoint code generation in a domain-specific IR
- TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems
- PyTorch: An Imperative Style, High-Performance Deep Learning Library
- <https://ai-techsystems.com/dnn-compiler/>

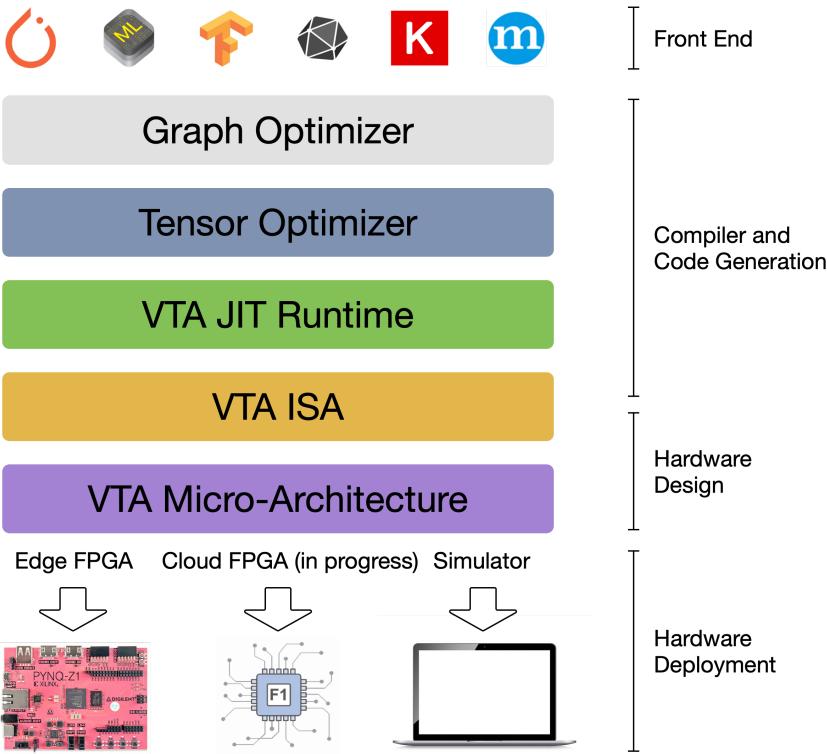
TVM and Versatile Tensor Accelerator (VTA)

TVM is an open deep learning compiler stack for CPUs, GPUs, and specialized accelerators. It aims to close the gap between the productivity-focused deep learning frameworks, and the performance- or efficiency-oriented hardware backends. TVM provides the following main features:

Compilation of deep learning models in Keras, MXNet, PyTorch, Tensorflow, CoreML, DarkNet into minimum deployable modules on diverse hardware backends. Infrastructure to automatically generate and optimize tensor operators on more backend with better performance.



The Versatile Tensor Accelerator (VTA) is an extension of the TVM framework designed to advance deep learning and hardware innovation. VTA is a programmable accelerator that exposes a RISC-like programming abstraction to describe compute and memory operations at the tensor level. We designed VTA to expose the most salient and common characteristics of mainstream deep learning accelerators, such as tensor operations, DMA load/stores, and explicit compute/memory arbitration.



- <https://sampl.cs.washington.edu/>
- <https://homes.cs.washington.edu/~haichen/>
- <https://tqchen.com/>
- <https://www.cs.washington.edu/people/faculty/arvind>
- [TVM: End to End Deep Learning Compiler Stack](#)
- [TVM and Deep Learning Compiler Conference](#)
- [VTA Deep Learning Accelerator](#)
- <https://sampl.cs.washington.edu/>
- <https://docs.tvm.ai/vta/index.html>
- [如何利用TVM快速实现超越Numpy\(MKL\)的GEMM蓝色](#)
- [使用TVM支持TFLite \(下\)](#)

nGraph

nGraph is an end to end deep learning compiler for inference and training with extensive framework and hardware support.

- <https://www.ngraph.ai/>
- <https://github.com/NervanaSystems/ngraph>
- <https://github.com/plaidml/plaidml>
- [https://en.wikipedia.org/wiki/Zeroth_\(software\)](https://en.wikipedia.org/wiki/Zeroth_(software))

XLA

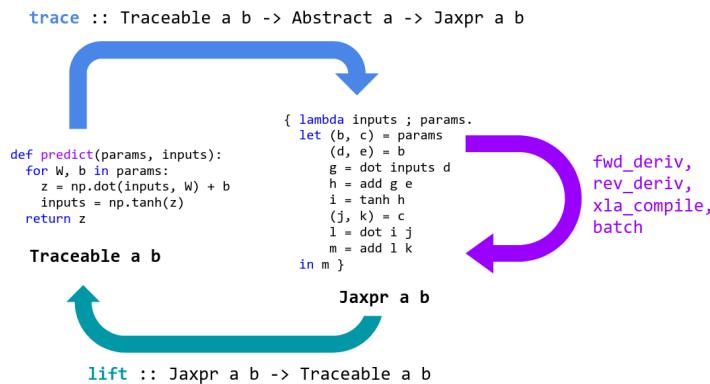
The XLA compilation framework is invoked on subgraphs of TensorFlow computations. The framework requires all tensor shapes to be fixed, so compiled code is specialized to concrete shapes. This means, for example, that the compiler may be invoked multiple times for the same subgraph if it is executed on batches of different sizes.

- <https://www.tensorflow.org/versions/master/experimental/xla/>
- <https://developers.googleblog.com/2017/03/xla-tensorflow-compiled.html>

- <https://www.tensorflow.org/xla/overview>
- <https://autodiff-workshop.github.io/slides/JeffDean.pdf>
- XLA: The TensorFlow compiler framework

JAX: Autograd and XLA

With its updated version of Autograd, JAX can automatically differentiate native Python and NumPy functions. It can differentiate through loops, branches, recursion, and closures, and it can take derivatives of derivatives of derivatives. It supports reverse-mode differentiation (a.k.a. backpropagation) via grad as well as forward-mode differentiation, and the two can be composed arbitrarily to any order.



- <https://github.com/google/jax>
- <https://github.com/hips/autograd>

Multi-Level Intermediate Representation

The Multi-Level Intermediate Representation (MLIR) is intended for easy expression and optimization of computations involving deep loop nests and dense matrices of high dimensionality. It is thus well-suited to deep learning computations in particular. Yet it is general enough to also represent arbitrary sequential computation. The representation allows high-level optimization and parallelization for a wide range of parallel architectures including those with deep memory hierarchies --- general-purpose multicores, GPUs, and specialized neural network accelerators.

- <https://github.com/tensorflow/mlir>
- <https://llvm.org/devmtg/2019-04/slides/Keynote-ShpeismanLattner-MLIR.pdf>

Glow

Glow is a machine learning compiler and execution engine for hardware accelerators. It is designed to be used as a backend for high-level machine learning frameworks. The compiler is designed to allow state of the art compiler optimizations and code generation of neural network graphs. This library is in active development.

- <https://arxiv.org/pdf/1805.00907.pdf>
- <https://ai.facebook.com/tools/glow/>
- <https://github.com/pytorch/glow>

Cilk

Cilk aims to make parallel programming a simple extension of ordinary serial programming. Other concurrency platforms, such as Intel's Threading Building Blocks (TBB) and OpenMP, share similar goals of making parallel programming easier. But Cilk sets itself apart from other

concurrency platforms through its simple design and implementation and its powerful suite of provably effective tools. These properties make Cilk well suited as a platform for next-generation multicore research. Tapir enables effective compiler optimization of parallel programs with only minor changes to existing compiler analyses and code transformations. Tapir uses the serial-projection property to order logically parallel fine-grained tasks in the program's control-flow graph. This ordered representation of parallel tasks allows the compiler to optimize parallel codes effectively with only minor modifications.

- <http://cilk.mit.edu/tapir/>
- <https://llvm.org/>
- <https://zhuanlan.zhihu.com/p/64903359>

Triton

- Triton: An Intermediate Language and Compiler for Tiled Neural Network Computations
- <https://github.com/ptillet/triton>
- <https://pldi19.sigplan.org/details/mapl-2019-papers/1/Triton-An-Intermediate-Language-and-Compiler-for-Tiled-Neural-Network-Computations>
- <http://www.federated-ml.org/tutorials/globecom2020/part4.pdf>

DLVM

- DLVM: A MODERN COMPILER INFRASTRUCTURE FOR DEEP LEARNING SYSTEMS
- <http://dowobeha.github.io/papers/autodiff17.pdf>
- <http://dowobeha.github.io/>
- <https://roshandathathri.github.io/>
- <https://www.clsp.jhu.edu/workshops/19-workshop/>

CHET

- <https://www.cs.utexas.edu/~roshan/CHET.pdf>
- <https://www.cs.utexas.edu/~pingali/>
- <https://roshandathathri.github.io/>
- <https://iss.oden.utexas.edu/>

NeDLR

DLR is a compact, common runtime for deep learning models and decision tree models compiled by AWS SageMaker Neo, TVM, or Treelite. DLR uses the TVM runtime, Treelite runtime, NVIDIA TensorRT™, and can include other hardware-specific runtimes. DLR provides unified Python/C++ APIs for loading and running compiled models on various devices. DLR currently supports platforms from Intel, NVIDIA, and ARM, with support for Xilinx, Cadence, and Qualcomm coming soon.

- <https://github.com/neo-ai/neo-ai-dlr>
- <https://github.com/dmlc/treelite>
- <http://treelite.io/>
- <https://aws.amazon.com/cn/sagemaker/neo/>

Butterflies: A Universal Building Block for Structured Linear Maps

Fast linear transforms are ubiquitous in machine learning, including the discrete Fourier transform, discrete cosine transform, and other structured transformations such as convolutions. All of these transforms can be represented by dense matrix-vector multiplication, yet each has a

specialized and highly efficient (subquadratic) algorithm. We ask to what extent hand-crafting these algorithms and implementations is necessary, what structural priors they encode, and how much knowledge is required to automatically learn a fast algorithm for a provided structured transform. Motivated by a characterization of `fast matrix-vector multiplication` as products of sparse matrices, we introduce a parameterization of divide-and-conquer methods that is capable of representing a large class of transforms. This generic formulation can automatically learn an efficient algorithm for many important transforms; for example, it recovers the $O(N \log N)$ Cooley-Tukey FFT algorithm to machine precision, for dimensions N up to 1024. Furthermore, our method can be incorporated as a lightweight replacement of generic matrices in machine learning pipelines to learn efficient and compressible transformations. On a standard task of compressing a single hidden-layer network, our method exceeds the classification accuracy of unconstrained matrices on CIFAR-10 by 3.9 points ---the first time a structured approach has done so---with 4X faster inference speed and 40X fewer parameters.

- [Learning Fast Algorithms for Linear Transforms Using Butterfly Factorizations](#)
- [Learning Fast Algorithms for Linear Transforms Using Butterfly Factorizations](#)
- [Butterflies Are All You Need: A Universal Building Block for Structured Linear Maps](#)
- <https://github.com/stanford-futuredata/Willump>
- [A Two Pronged Progress in Structured Dense Matrix Multiplication](#)

halide

Halide is a programming language designed to make it easier to write high-performance image and array processing code on modern machines.

- <https://halide-lang.org/>
- <https://halide-lang.org/#publications>
- <https://people.csail.mit.edu/tzumao/>

taichi

Taichi (太极) is a programming language designed for high-performance computer graphics. It is deeply embedded in Python, and its just-in-time compiler offloads compute-intensive tasks to multi-core CPUs and massively parallel GPUs

- <http://taichi.graphics/>
- <https://github.com/yuanming-hu/taichi>
- http://taichi.graphics/wp-content/uploads/2019/09/taichi_lang.pdf

TASO

TASO optimizes the computation graphs of DNN models using automatically generated and verified graph transformations. For an arbitrary DNN model, TASO uses the auto-generated graph transformations to build a large search space of potential computation graphs that are equivalent to the original DNN model. TASO employs a cost-based search algorithm to explore the space, and automatically discovers highly optimized computation graphs.

- <https://cs.stanford.edu/people/zhihao/>
- <https://github.com/jiazhihao/TASO>
- <http://theory.stanford.edu/~aiken/>

Numerical algorithms for high-performance computational science

Several key themes emerged across multiple talks in [Royal Society Discussion Meeting](#), all in the context of today's high performance computing landscape in which processor clock speeds have stagnated (with the end of Moore's law) and exascale machine are just two or three years away.

- An important way of accelerating computations is through the use of low precision floating-point arithmetic—in particular by exploiting a hierarchy of precisions.
- We must exploit low rank matrix structure where it exists, for example in hierarchical (H-matrix) form, combining it with randomized approximations.
- Minimizing data movement (communication) is crucial, because of its increasing costs relative to the costs of floating-point arithmetic.
- Co-design (the collaborative and concurrent development of hardware, software, and numerical algorithms, with knowledge of applications) is increasingly important for numerical computing.

For more on high performance computation on GPU see <https://hgpu.org/>.

- <https://hgpu.org/>
- Numerical Algorithms for High-Performance Computational Science: Highlights of the Meeting
- Numerical algorithms for high-performance computational science
- Reflections on the Royal Society's "Numerical Algorithms for High-performance Computational Science" Discussion Meeting
- Overview of Microsoft HPC Pack 2016
- Document Library: High Performance Computing Fabrics
- MVAPICH: MPI over InfiniBand, Omni-Path, Ethernet/iWARP, and RoCE
- <https://researchcomputing.lehigh.edu/>
- <https://library.columbia.edu/libraries/dsc/hpc.html>
- Open MPI: Open Source High Performance Computing
- <https://ltsnews.lehigh.edu/node/115>
- NumFOCUS
- <http://www.mit.edu/~kepner/D4M/>
- CSCS-ICS-DADSi Summer School: Accelerating Data Science with HPC, September 4 – 6, 2017
- MS&E 317: Algorithms for Modern Data Models: Spring 2014, Stanford University
- Distributed Machine Learning and Matrix Computations: A NIPS 2014 Workshop
- Large Scale Matrix Analysis and Inference: A NIPS 2013 Workshop
- Breakthrough! Faster Matrix Multiply
- Performance Engineering of Software Systems
- <https://zhuanlan.zhihu.com/p/94653447>

Why GEMM is at the heart of deep learning

General Matrix Multiply (GEMM) is a common algorithm in linear algebra, machine learning, statistics, and many other domains.

- <https://github.com/pytorch/QNNPACK>

- <https://github.com/XiaoMi/mace>

Fast Matrix-vector Multiplication

Matrix-vector multiplication is a special matrix multiplication:

$$\mathbb{R}^m \mapsto \mathbb{R}^n : Mv \rightarrow u$$

$$u = Mv = \sum_{i=1} A^{(i)} v_i$$

where $M \in \mathbb{R}^{m \times n}$, $u \in \mathbb{R}^n$; each column $M^{(i)}$ can, metaphorically, indicate one address or house and each $v(i)$ a letter addressed to it.

- <https://people.csail.mit.edu/rrw/>
- Matrix-Vector Multiplication in Sub-Quadratic Time (Some Preprocessing Required)
- The Mailman algorithm for matrix vector multiplication
- Matrix-vector multiplication using the FFT
- On fast matrix-vector multiplication with a Hankel matrix in multiprecision arithmetics
- Optimizing Large Matrix-Vector Multiplications
- Faster Matrix vector multiplication ON GeForce
- Fast Implementation of General Matrix-Vector Multiplication (GEMV) on Kepler GPUs
- FAST ALGORITHMS TO COMPUTE MATRIX-VECTOR PRODUCTS FOR PASCAL MATRICES
- Fast algorithms to compute matrix-vector products for Toeplitz and Hankel matrices
- Fast High Dimensional Vector Multiplication Face Recognition
- Faster Online Matrix-Vector Multiplication
- A fast matrix–vector multiplication method for solving the radiosity equation
- Computational Science and Engineering Spring 2015 syllabus

Computation of Matrix Chain Products

Generations of students have learned that the product $xy^T z$, where x , y , and z are n-vectors, should be written and evaluated as $x(y^T z)$ ($O(n)$ flops) rather than $(xy^T)z$ ($O(n^2)$) flops). More generally, deciding where to put the parentheses in a matrix product $A_1 A_2 \dots A_k$ to minimize the number of operations in the evaluation is a nontrivial problem, known as the **matrix chain multiplication problem**.

A special case is when $A_1 = A_2 = \dots = A_k$ the problem is to compute the $A^k = \underbrace{A \cdots A}_k$.

- The World's Most Fundamental Matrix Equation
- Computation of Matrix Chain Products, PART I, PART II
- CS3343/3341 Analysis of Algorithms Matrix-chain Multiplications
- rosettacode: Matrix chain multiplication
- <http://www.columbia.edu/~cs2035/courses/csor4231.F11/matrix-chain.pdf>
- <https://www.geeksforgeeks.org/matrix-chain-multiplication-dp-8/>
- <https://www.geeksforgeeks.org/matrix-chain-multiplication-a-on2-solution/>
- <https://home.cse.ust.hk/~dehai/271/notes/L12/L12.pdf>

Generalized Matrix to Matrix Multiplication

If the computation speed of matrix operation is boosted, the inference of deep learning model is accelerated. Matrix multiplication $C_{M \times N} = A_{M \times K} B_{K \times N}$ via dot product is defined as

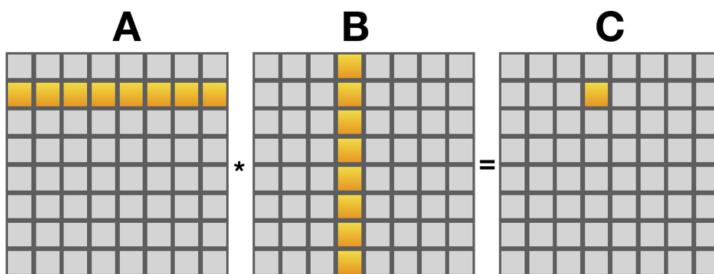
$$C[m, n] = \langle A[m, :], B[:, n] \rangle = \sum_{k=1}^K A[m, k] \times B[k, n]$$

which is essentially product-sum.

```
for (int m = 0; m < M; m++) {
    for (int n = 0; n < N; n++) {
        C[m][n] = 0;
        for (int k = 0; k < K; k++) {
            C[m][n] += A[m][k] * B[k][n];
        }
    }
}
```

It needs $O(MKN)$ multiplication.

The picture below visualizes the computation of a single element in the result matrix C . Each element in the result matrix C is the sum of element-wise multiplication of a row from A and a column from B .



Our program is memory bound, which means that the multipliers are not active most of the time because they are waiting for memory.

- [Anatomy of High-Performance Matrix Multiplication](#)
- [Geometry and the complexity of matrix multiplication](#)
- [High-Performance Matrix Multiplication](#)
- [Fast Matrix Multiplication @mathoverflow](#)
- [Powers of Tensors and Fast Matrix Multiplication](#)
- <https://www.kkhaydarov.com/matrix-multiplication-algorithms/>
- [BLISlab: A Sandbox for Optimizing GEMM](#)
- [MAGMA: Matrix Algebra for GPU and Multicore Architectures](#)
- <http://apfel.mathematik.uni-ulm.de/~lehn/sghpc/gemm/>
- [通用矩阵乘和卷积优化](#)
- [Fast Matrix Multiplication Algorithms](#)
- [Anatomy of high-performance matrix multiplication](#)
- [The Indirect Convolution Algorithm](#)
- <https://github.com/flame/how-to-optimize-gemm/wiki>
- https://en.wikipedia.org/wiki/Matrix_multiplication#Complexity
- [Matrix multiplication via arithmetic progressions](#)
- [Fast sparse matrix multiplication](#)
- [Part I: Performance of Matrix multiplication in Python, Java and C++](#)
- [Part III: Matrix multiplication on multiple cores in Python, Java and C++](#)

- <https://github.com/MartinThoma/matrix-multiplication>
- <http://jianyuhuang.com/>
- SPATIAL: A high-level language for programming accelerators
- <https://github.com/OAID/Tengine>
- Performance of Classic Matrix Multiplication Algorithm on Intel® Xeon Phi™ Processor System
- Low-precision matrix multiplication

Strassen Algorithms

It is based on block-multiplication. It is required that $C \in \mathbb{R}^{2^n \times 2^n}$.

The matrices are rearranged as blocks:

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{1,1} & \mathbf{A}_{1,2} \\ \mathbf{A}_{2,1} & \mathbf{A}_{2,2} \end{bmatrix}, \mathbf{B} = \begin{bmatrix} \mathbf{B}_{1,1} & \mathbf{B}_{1,2} \\ \mathbf{B}_{2,1} & \mathbf{B}_{2,2} \end{bmatrix},$$

$$\mathbf{C} = \begin{bmatrix} \mathbf{C}_{1,1} & \mathbf{C}_{1,2} \\ \mathbf{C}_{2,1} & \mathbf{C}_{2,2} \end{bmatrix} = \begin{bmatrix} \mathbf{A}_{1,1}\mathbf{B}_{1,1} & \mathbf{A}_{1,2}\mathbf{B}_{1,2} \\ \mathbf{A}_{2,1}\mathbf{B}_{2,1} & \mathbf{A}_{2,2}\mathbf{B}_{2,2} \end{bmatrix}.$$

Submatrix(blocks) multiplication is performed in the following way:

$$\begin{aligned} \mathbf{M}_1 &= (\mathbf{A}_{1,1} + \mathbf{A}_{2,2})(\mathbf{B}_{1,1} + \mathbf{B}_{2,2}) \\ \mathbf{M}_2 &= (\mathbf{A}_{2,1} + \mathbf{A}_{2,2})\mathbf{B}_{1,1} \\ \mathbf{M}_3 &= \mathbf{A}_{1,1}(\mathbf{B}_{1,2} - \mathbf{B}_{2,2}) \\ \mathbf{M}_4 &= \mathbf{A}_{1,2}(\mathbf{B}_{2,1} - \mathbf{B}_{1,1}) \\ \mathbf{M}_5 &= (\mathbf{A}_{1,1} + \mathbf{A}_{1,2})\mathbf{B}_{2,2} \\ \mathbf{M}_6 &= (\mathbf{A}_{2,1} - \mathbf{A}_{1,1})(\mathbf{B}_{1,1} + \mathbf{B}_{1,2}) \\ \mathbf{M}_7 &= (\mathbf{A}_{1,2} - \mathbf{A}_{2,2})(\mathbf{B}_{2,1} + \mathbf{B}_{2,2}) \end{aligned}$$

And then

$$\begin{aligned} \mathbf{C}_{1,1} &= \mathbf{M}_1 + \mathbf{M}_4 - \mathbf{M}_5 + \mathbf{M}_7 \\ \mathbf{C}_{1,2} &= \mathbf{M}_3 + \mathbf{M}_5 \\ \mathbf{C}_{2,1} &= \mathbf{M}_2 + \mathbf{M}_4 \\ \mathbf{C}_{2,2} &= \mathbf{M}_1 - \mathbf{M}_2 + \mathbf{M}_3 + \mathbf{M}_6 \end{aligned}$$

- Strassen's Matrix Multiplication Algorithm | Implementation
- Part II: The Strassen algorithm in Python, Java and C++
- Using Strassen's Algorithm to Accelerate the Solution of Linear System
- <https://shivathudi.github.io/jekyll/update/2017/06/15/matr-mult.html>
- <http://jianyuhuang.com/papers/sc16.pdf>
- Comparative Study of Strassen's Matrix Multiplication Algorithm
- <http://andrew.gibiansky.com/blog/mathematics/matrix-multiplication/>
- <http://jianyuhuang.com/>

Coppersmith–Winograd Algorithms

For matrix multiplication $U = V = W = \mathbb{F}^{n \times n}$, and we write this bilinear map as $\phi = \langle \cdot, \cdot \rangle$ where $\phi(\cdot, \cdot) : U \times V \mapsto W$ is bilinear.

The tensor corresponding to the multiplication of an $m \times n$ matrix by an $n \times p$ matrix is

$$\langle m, n, p \rangle = \sum_{i=1}^m \sum_{j=1}^p \sum_{k=1}^n a_{ik} \otimes b_{kj} \otimes c_{ij}.$$

One can define in a natural way the tensor product of two tensors. In particular, for matrix multiplication tensors, we obtain the following identity: for any positive integers m, m_0, n, n_0, p, p_0 ,

$$\langle m, n, p \rangle \otimes \langle m_0, n_0, p_0 \rangle = \langle mm_0, nn_0, pp_0 \rangle.$$

Consider three vector spaces U, V and W over the field \mathbb{F} and

- $U = \text{span}\{x_1, \dots, x_{\dim(U)}\}$,
- $V = \text{span}\{y_1, \dots, y_{\dim(V)}\}$,
- $W = \text{span}\{z_1, \dots, z_{\dim(W)}\}$.

A tensor over (U, V, W) is an element of $U \otimes V \otimes W$ i.e., a formal sum

$$T = \sum_{u=1}^{\dim(U)} \sum_{v=1}^{\dim(V)} \sum_{w=1}^{\dim(W)} \underbrace{d_{uvw}}_{\in \mathbb{F}} x_u \otimes y_v \otimes z_w.$$

We use tensors (and their low-rank decompositions) to multiply matrices faster than $O(n^3)$.

Define the matrix multiplication tensor as follows:

$$M_{(a,b),(c,d),(e,f)}^{(n)} = \begin{cases} 1, & \text{if } b = c, d = e, f = a, \\ 0, & \text{otherwise.} \end{cases}$$

Suppose $\text{rank}(M^{(n)}) \leq r$, i.e.,

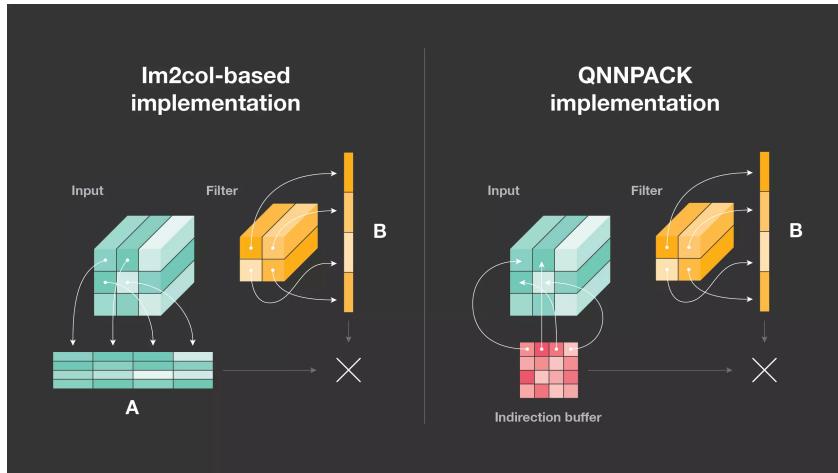
$$M_{(a,b),(c,d),(e,f)}^{(n)} = \sum_{\ell=1}^r x_{ab}^\ell y_{cd}^\ell z_{ef}^\ell.$$

Then we can use this decomposition to re-express matrix multiplication:

$$\begin{aligned} (AB)_{ik} &= \sum_j A_{ij} B_{jk} \\ &= \sum_j M_{(i,j),(j,k),(k,i)}^{(n)} A_{ij} B_{jk} \\ &= \sum_{a=1}^n \sum_{b=1}^n \sum_{c=1}^n \sum_{d=1}^n M_{(a,b),(c,d),(k,i)}^{(n)} A_{ab} B_{cd} \\ &= \sum_{a=1}^r \sum_{b=1}^n \sum_{c=1}^n \sum_{d=1}^n \left(\sum_{l=1}^r x_{ab}^l y_{cd}^l z_{ki}^l \right) A_{ab} B_{cd} \\ &= \sum_{l=1}^r z_{ki}^l \left(\sum_{a=1}^n \sum_{b=1}^n x_{ab}^l A_{ab} \right) \left(\sum_{c=1}^n \sum_{d=1}^n y_{cd}^l B_{cd} \right) \end{aligned}$$

- Fast Matrix Multiplication = Calculating Tensor Rank, Caltech Math 10 Presentation
- <http://users.wfu.edu/ballard/pdfs/CSE17.pdf>
- <https://jackwish.net/reveal-qnnpack-implementation.html>
- Coppersmith–Winograd Algorithm
- On the Coppersmith–Winograd method
- Adaptive Winograd’s Matrix Multiplications
- https://www.wikiwand.com/en/Matrix_multiplication_algorithm
- Introduction to fast matrix multiplication

- Practical Fast Matrix Multiplication Algorithms



- Breaking the Coppersmith-Winograd barrier
- Multiplying matrices faster than Coppersmith-Winograd
- Limits on All Known (and Some Unknown) Approaches to Matrix Multiplication
- A New Fast Recursive Matrix Multiplication Algorithm
- <https://cqa.institute/2015/08/07/accelerating-matrix-multiplication/>
- Distinguished Lecture Series III: Avi Wigderson, "Algebraic computation"
- <http://www.cs.utexas.edu/~rvdg/>

Linear Algebra Packages

- Halide: a language for fast, portable computation on images and tensors
- <https://eigen.tuxfamily.org/dox/index.html>
- FLAME project
- A high-level language for programming accelerators
- <http://people.ece.umn.edu/users/parhi/>
- <https://www.cs.utexas.edu/~flame/web/publications.html>
- PLAPACK: Parallel Linear Algebra Package
- <https://spatial-lang.readthedocs.io/en/latest/>
- Matrix Computations and Optimization in Apache Spark
- Cusp is a library for sparse linear algebra and graph computations based on Thrust.
- The Hierarchical Computations on Manycore Architectures (HiCMA) library aims to redesign existing dense linear algebra libraries to exploit the data sparsity of the matrix operator.
- A generalized multidimensional matrix multiplication
- <https://dspace.mit.edu/handle/1721.1/85943>
- BLIS Retreat 2017

Automatic Differentiation, Differentiable Programming and Program Transformations

Automatic Differentiation

All numerical gradient-based optimization methods benefits from faster computation of gradients specially backprop.

Many algorithms in machine learning, computer vision, physical simulation, and other fields require the calculation of gradients and other derivatives. Manual derivation of gradients can be time consuming and error-prone. Automatic Differentiation

(AD) is a technology for automatically augmenting computer programs, including arbitrarily complex simulations, with statements for the computation of derivatives, also known as sensitivities. Automatic differentiation comprises a set of techniques to calculate the derivative of a numerical computation expressed as a computer program. These techniques are commonly used in atmospheric sciences and computational fluid dynamics, and have more recently also been adopted by machine learning researchers. Practitioners across many fields have built a wide set of automatic differentiation tools, using different programming languages, computational primitives and intermediate compiler representations. Each of these choices comes with positive and negative trade-offs, in terms of their usability, flexibility and performance in specific domains.

In the ideal case, automatically generated derivatives should be competitive with manually generated ones and run at near-peak performance on modern hardware, but the most expressive systems for autodiff which can handle arbitrary, Turing-complete programs, are unsuited for performance-critical applications, such as large-scale machine learning or physical simulation. [Alternatively, the most performant systems are not designed for use outside of their designated application space, e.g. graphics or neural networks.](#)

- <https://autodiff-workshop.github.io/>
- <https://autodiff-workshop.github.io/2016.html>
- <http://www.autodiff.org/>
- [Tools for Automatic Differentiation](#)
- <https://arxiv.org/abs/1611.01652>
- [The simple essence of automatic differentiation](#)

"What does AD mean, independently of implementation?" An answer arises in the form of naturality of sampling a function and its derivative. Automatic differentiation flows out of this naturality condition, together with the chain rule. Graduating from first-order to higher-order AD corresponds to sampling all derivatives instead of just one. Next, the setting is expanded to arbitrary vector spaces, in which derivative values are linear maps. [The specification of AD adapts to this elegant and very general setting, which even simplifies the development.](#)

- [autodiff is a C++17 library for automatic computation of derivatives](#)
- [DiffSharp: Differentiable Functional Programming](#)
- <https://www.mcs.anl.gov/OpenAD/>
- <https://github.com/google/tangent>
- https://en.wikipedia.org/wiki/Automatic_differentiation
- <http://www.admb-project.org/>
- <https://github.com/rjhogan/Adept>
- <https://fluxml.ai/Zygote.jl/latest/>
- <http://www.met.reading.ac.uk/clouds/adept/>
- [AD computation with Template Model Builder \(TMB\)](#)
- <https://www.juliadiff.org/>
- [autodiffr for Automatic Differentiation in R through Julia](#)
- <https://srijithr.gitlab.io/post/autodiff/>
- <https://fl.readthedocs.io/en/latest/autograd.html>
- <https://pymanopt.github.io/>
- <https://yiduai.sg/tensorflow-workshop/>

- Swift
- <https://github.com/Functional-AutoDiff/STALINGRAD>
- <https://coin-or.github.io/CppAD/doc/cppad.htm>
- <http://simweb.iwr.uni-heidelberg.de/~darndt/files/doxygen/deal.II/index.html>
- AD-Suite: A Test Suite for Algorithmic Differentiation

Differentiable Programming

Deep learning may look like another passing fad, in the vein of "expert systems" or "big data." But it's based on two timeless ideas (back-propagation and weight-tying), and while differentiable programming is a very new concept, it's a natural extension of these ideas that may prove timeless itself. Even as specific implementations, architectures, and technical phrases go in and out of fashion, [these core concepts will continue to be essential to the success of AI.](#)

Constructing neural networks using pure and higher-order differentiable functions and training them using reverse-mode automatic differentiation is unsurprisingly called **Differentiable Programming**.

- What Is Differentiable Programming?
- <https://www.lokad.com/differentiable-programming>
- Google Summer of Code Projects
- Flux: The Julia Machine Learning Library
- DiffEqFlux.jl – A Julia Library for Neural Differential Equations
- Demystifying Differentiable Programming: Shift/Reset the Penultimate Backpropagator
- Diòerentiable Visual Computing by Tzu-Mao Li

---	Deep Learning	Differentiable Programming
Primary purpose	Learning	Learning+Optimization
Typical usage	Learn-once, Eval-many	Learn-once, Eval-once
Input granularity	Fat objects (images, voice sequences, lidar scans, full text pages)	Thin objects (products, clients, SKUs, prices)
Input variety	Homogeneous objects (e.g. images all having the same height/width ratio)	Heterogeneous objects (relational tables, graphs, time-series)
	<ul style="list-style-type: none"> • Probabilistic & Differentiable Programming Summit • Differentiable Programming for Image Processing and Deep Learning in Halide • https://github.com/sunze1/Differential-Programming • Differentiable Programming: A Semantics Perspective • https://fixpointsandcoffee.com/computer-science/169/ • Zygote: A Differentiable Programming System to Bridge Machine Learning and Scientific Computing • Differentiable Programming for Image Processing and Deep Learning in Halide • https://github.com/Hananel-Hazan/bindsnet • https://skymind.ai/wiki/differentiableprogramming • https://people.csail.mit.edu/tzumao/ 	

- <https://people.eecs.berkeley.edu/~jrk/>
- <http://people.csail.mit.edu/fredo/>

Program Transformations

[Program Transformations for Machine Learning](#) - Workshop at NeurIPS 2019 – December 13 or 14 2019, Vancouver, Canada - claims that

Machine learning researchers often express complex models as a program, relying on program transformations to add functionality. New languages and transformations (e.g., TorchScript and TensorFlow AutoGraph) are becoming core capabilities of ML libraries. However, existing transformations, such as `automatic differentiation` (AD or autodiff), inference in `probabilistic programming languages` (PPLs), and `optimizing compilers` are often built in isolation, and limited in scope. This workshop aims at viewing program transformations in ML in a unified light, making these capabilities more accessible, and building entirely new ones.

Program transformations are an area of active study. AD transforms a program performing numerical computation into one computing the gradient of those computations. In probabilistic programming, a program describing a sampling procedure can be modified to perform inference on model parameters given observations. Other examples are vectorizing a program expressed on one data point, and learned transformations where ML models use programs as inputs or outputs.

This workshop will bring together researchers in the fields of AD, probabilistic programming, programming languages, compilers, and ML with the goal of understanding the commonalities between disparate approaches and views, and sharing ways to make these techniques broadly available. It would enable ML practitioners to iterate faster on novel models and architectures (e.g., those naturally expressed through high-level constructs like recursion).

- <https://popl19.sigplan.org/track/lafi-2019#About>
- <https://program-transformations.github.io/>
- <https://uncertainties-python-package.readthedocs.io/en/latest/>
- <https://conf.researchr.org/track/POPL-2017/pps-2017>
- <https://gustavoasoares.github.io/>
- <https://kedar-namjoshi.github.io/>
- Learning Syntactic Program Transformations from Examples
- <https://alexpologozov.com/>
- <https://kedar-namjoshi.github.io/>
- <https://vega.github.io/vega/>

Fixed-point Arithmetic and Approximate Computing

Today's computing systems are designed to deliver only exact solutions at high energy cost, while many of the algorithms that are run on data are at their heart statistical, and thus do not require exact answers.

It turns out that it is sometimes possible to get high-accuracy solutions from low-precision training—and here we'll describe a new variant of stochastic gradient descent (SGD) called high-accuracy low precision (HALP) that can do it. HALP can do better than previous algorithms

because it reduces the two sources of noise that limit the accuracy of low-precision SGD: gradient variance and round-off error.

- https://www.wikiwand.com/en/Fixed-point_arithmetic
- Approximate Computing
- PACO: The Worlds First Approximate Computing General Purpose CPU
- System Energy Efficiency Lab
- <http://www.oliviervalery.com/publications/pdp2018>
- <https://devblogs.nvidia.com/int8-inference-autonomous-vehicles-tensorrt/>
- <https://nvidia.github.io/apex/>
- A Multiprecision World
- <https://arxiv.org/abs/1806.00875v1>

High-Performance Deep Learning

- High Performance Distributed Deep Learning
- High-Performance Deep Learning (HiDL)
- High Performance Distributed Deep Learning: A Beginner's Guide
- <http://prace.it4i.cz/en/HPDDL-01-2018>
- <https://github.com/drcut/CPD>
- <https://dlonsc19.github.io/>
- <http://prace.it4i.cz/en/HPDDL-01-2018>

Compression and Acceleration of Convolutional Neural Networks

CNN is the most wisely used deep learning models in computer vision.

Theme Name	Description	Application	More Details
Parameter pruning and sharing	Reducing redundant parameters which are not sensitive to the performance	Convolutional layer and fully connected layer	Robust to various setting, can achieve good performance, can support both train from scratch and pre-trained model
Low-rank factorization	Using matrix/tensor decomposition to estimate the information parameters	Convolutional layer and fully connected layer	Standardized pipeline, easily to be implemented, can support both train from scratch and pre-trained model
Transferred/compact convolutional filters	Designing special structural convolutional filter to save parameters	Convolutional layer only	Algorithms are dependent on applications, usually achieve good performance, only support train from scratch

Theme Name	Description	Application	More Details
Knowledge distillation	Training a compact neural network with distilled knowledge of a large model	Convolutional layer and fully connected layer	Model performances are sensitive to applications and network structure only support train from scratch

- Stanford Compression Forum
- <https://jackwish.net/convolution-neural-networks-optimization.html>
- Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks
- 深度学习如何进行模型压缩?
- Caffeine: Towards Uniformed Representation and Acceleration for Deep Convolutional Neural Networks
- Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks
- Automated Systolic Array Architecture Synthesis for High Throughput CNN Inference on FPGAs
- Efficient Deep Learning for Computer Vision CVPR 2019
- Model Compression and Acceleration for Deep Neural Networks: The Principles, Progress, and Challenges
- <https://www.ibm.com/blogs/research/2018/02/deep-learning-training/>

Parameter Pruning and Sharing

Pruning is to prune the connections in deep neural network in order to reduce the number of weights.

- Learn the connectivity via normal network training
- Prune the low-weight connections
- Retrain the sparse network

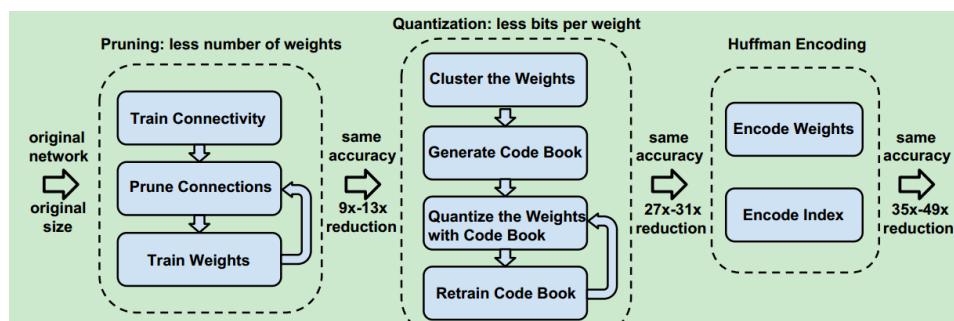
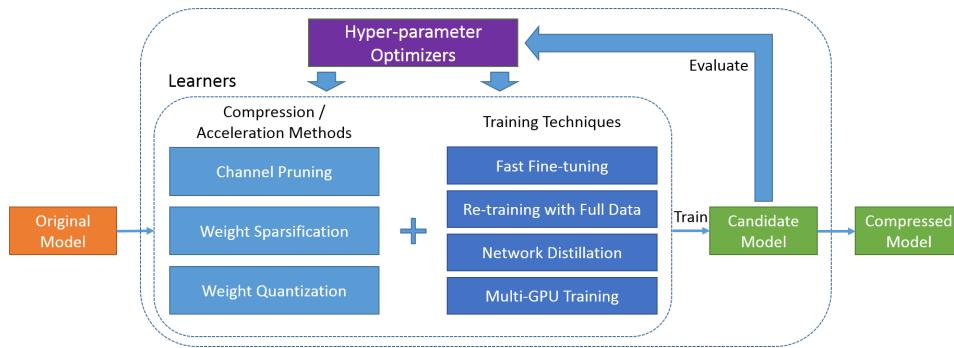


Figure 1: The three stage compression pipeline: pruning, quantization and Huffman coding. Pruning reduces the number of weights by $10\times$, while quantization further improves the compression rate: between $27\times$ and $31\times$. Huffman coding gives more compression: between $35\times$ and $49\times$. The compression rate already included the meta-data for sparse representation. The compression scheme doesn't incur any accuracy loss.

- Pruning deep neural networks to make them fast and small
- <https://nervanasystems.github.io/distiller/pruning/index.html>
- <https://github.com/yihui-he/channel-pruning>
- https://pocketflow.github.io/cp_learner/
- A Systematic DNN Weight Pruning Framework using Alternating Direction Method of Multipliers



Quantization and Fixed-point Arithmetic

Network quantization compresses the original network by reducing the number of bits required to represent each weight.

Uniform quantization is widely used for model compression and acceleration. Originally the weights in the network are represented by 32-bit floating-point numbers. With uniform quantization, low-precision (e.g. 4-bit or 8-bit) fixed-point numbers are used to approximate the full-precision network. For k -bit quantization, the memory saving can be up to $32/k$. For example, 8-bit quantization can reduce the network size by 4 folds with negligible drop of performance. The l th quantized ReLU $\sigma(x_l, \alpha_l)$ acts element-wise on vector x_l from a previous layer and is parameterized by trainable scalar $\alpha_l > 0$. In uniform quantization,

$$\sigma(x, \alpha) = \begin{cases} 0, & \text{if } x \leq 0, \\ k\alpha, & \text{if } (k-1)\alpha < x \leq k\alpha, \quad k = 1, 2, \dots, 2^{b_a} - 1, \\ (2^{b_a} - 1)\alpha, & \text{if } x > (2^{b_a} - 1)\alpha, \end{cases} \quad (1)$$

where x is the scalar input, b_a is the bit-width, and k is the quantization level. For a 4-bit quantization, $b_a = 4$ and $2^{b_a} = 16$ levels exist, including zero.

Given a pre-defined full-precision model, the learner inserts quantization nodes and operations into the computation graph of the model. With activation quantization enabled, quantization nodes will also be placed after activation operations (e.g. ReLU).

In the training phase, both full-precision and quantized weights are kept. In the forward pass, quantized weights are obtained by applying the quantization function on full-precision weights. To update full-precision weights in the backward pass, since gradients w.r.t. quantized weights are zeros almost everywhere, we use the straight-through estimator (STE, Bengio et al., 2015) to pass gradients of quantized weights directly to full-precision weights for update.

- [Quantized Neural Network PACKage - mobile-optimized implementation of quantized neural network operators](#)
- [TensorQuant: A TensorFlow toolbox for Deep Neural Network Quantization](#)
- <https://dominikfhg.github.io/TensorQuant/>
- <https://zhuanlan.zhihu.com/p/38328685>
- [Distiller is an open-source Python package for neural network compression research.](#)
- [Neural Network Quantization](#)
- [Neural Network Quantization Resources](#)
- [FINN-R: An End-to-End Deep-Learning Framework for Fast Exploration of Quantized Neural Networks](#)
- [Making Neural Nets Work With Low Precision](#)

- Differentiable Soft Quantization: Bridging Full-Precision and Low-Bit Neural Networks
- Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference
- Lower Numerical Precision Deep Learning Inference and Training

Fixed-point Arithmetic

The precision of a fixed-point number is the number of digits to the right of the decimal point, and it normally stays the same when computations are performed on the number.

- A Fixed-Point Arithmetic Package
- <http://hackage.haskell.org/package/fixed-point>
- <https://courses.cs.washington.edu/courses/cse467/08au/labs/l5/fp.pdf>
- Fixed Point Arithmetic and Tricks

Low Bit Neural Network

- Convergence of a Relaxed Variable Splitting Coarse Gradient Descent Method for Learning Sparse Weight Binarized Activation Neural Network
- <https://www.math.uci.edu/~jxin/xue.pdf>
- <https://arxiv.org/abs/1711.07354v1>
- <https://www.math.uci.edu/~jxin/signals.html>
- Extremely Low Bit Neural Network: Squeeze the Last Bit Out with ADMM
- Toward Extremely Low Bit and Lossless Accuracy in DNNs with Progressive ADMM

Deep Model Compression

- <https://github.com/he-y/Awesome-Pruning>
- <https://www.microsoft.com/en-us/research/search/?q=Deep+Compression>
- <https://www.tinyml.org/summit/slides/tinyMLSummit2020-4-1-Choi.pdf>
- https://openaccess.thecvf.com/content_cvpr_2017/papers/Yu_On_Compressing_Deep_CVPR_2017_paper.pdf
- <https://github.com/mightydeveloper/Deep-Compression-PyTorch>
- <https://www.oki.com/en/rd/tt/dl/>
- Deep Compression, DSD Training and EIE: Deep Neural Network Model Compression, Regularization and Hardware Acceleration

8-bit-training

The state-of-the-art hardware platforms for training Deep Neural Networks (DNNs) are moving from traditional single precision (32-bit) computations towards 16 bits of precision -- in large part due to the high energy efficiency and smaller bit storage associated with using reduced-precision representations. However, unlike inference, training with numbers represented with less than 16 bits has been challenging due to the need to maintain fidelity of the gradient computations during back-propagation. Here we demonstrate, for the first time, the successful training of DNNs using 8-bit floating point numbers while fully maintaining the accuracy on a spectrum of Deep Learning models and datasets. In addition to reducing the data and computation precision to 8 bits, we also successfully reduce the arithmetic precision for additions (used in partial product accumulation and weight updates) from 32 bits to 16 bits through the introduction of a number of key ideas including chunk-based accumulation and floating point stochastic rounding.

- 8-Bit Precision for Training Deep Learning Systems
- Extending 8-bit training breakthroughs to the toughest deep learning models
- Hybrid 8-bit Floating Point (HFP8) Training and Inference for Deep Neural Networks
- Training Deep Neural Networks with 8-bit Floating Point Numbers
- <https://www.ibm.com/blogs/research/author/xiao-sun/>

Binarized Neural Network, Ternary Weight Networks, XOR-Net

Binarized Neural Network

Binary neural networks are networks with binary weights and activations at run time. At training time these weights and activations are used for computing gradients; however, the gradients and true weights are stored in full precision. [This procedure allows us to effectively train a network on systems with fewer resources.](#)

Forward Binarization

For forward propagation, we need two binary matrices; we thus binarize the weight matrix and the incoming activation from the previous layer.

$$\begin{array}{|c|c|c|} \hline -0.4 & -0.4 & 0.9 \\ \hline 0.9 & 0.4 & 0.8 \\ \hline 0.4 & -0.4 & -0.4 \\ \hline \end{array} \approx 0.2 \quad \begin{array}{|c|c|c|} \hline -1 & -1 & 1 \\ \hline 1 & 1 & 1 \\ \hline 1 & -1 & -1 \\ \hline \end{array} \quad W^B = \text{sign}(W) \quad \alpha = \min\|W\|_1$$

{1.1. Forward propagation:}

for $k = 1$ to L **do**

$$W_k^b \leftarrow \text{Binarize}(W_k)$$

$$s_k \leftarrow a_{k-1}^b W_k^b$$

$$a_k \leftarrow \text{BatchNorm}(s_k, \theta_k)$$

if $k < L$ **then**

$$a_k^b \leftarrow \text{Binarize}(a_k)$$

end if

end for

A key to the success of BNNs is the binary activation function, which clamps all negative inputs to -1 and all positive inputs to 1 . There are two binarized functions:

deterministic

stochastic

$$x^b = \text{Sign}(x) = \begin{cases} +1 & \text{if } x \geq 0, \\ -1 & \text{otherwise,} \end{cases} \quad x^b = \begin{cases} +1 & \text{with probability } p = \sigma(x), \\ -1 & \text{with probability } 1 - p, \end{cases}$$

Here $\sigma(x)$ is the “hard sigmoid” function: $\sigma(x) = \max(0, \min(1, \frac{x+1}{2}))$. The stochastic binarization is better than the Sign function but is harder to implement. As a result, the deterministic Sign function is used more often.

- <https://mohitjain.me/2018/07/14/bnn/>

Gradient Propagation Through Discretization

The derivative of the sign function is zero almost everywhere, making it incompatible with backpropagation. Thus, a straight-through estimator is used. This preserves the gradient's information and cancels large gradients.

-0.4	-0.4	1.4
1.2	0.4	0.8
0.4	-0.4	-0.4

≈

-0.4	-0.4	0
0	0.4	0.8
0.4	-0.4	-0.4

\mathbf{g}_q \mathbf{g}_r

$$\mathbf{g}_r = \mathbf{g}_q \mathbf{1}_{|\mathbf{r}| \leq 1}$$

{1.2. Backward propagation:}

{Please note that the gradients are not binary.}

Compute $g_{a_L} = \frac{\partial C}{\partial a_L}$ knowing a_L and a^*

for $k = L$ to 1 **do**

if $k < L$ **then**

$g_{a_k} \leftarrow g_{a_k^b} \circ \mathbf{1}_{|a_k| \leq 1}$

end if

$(g_{s_k}, g_{\theta_k}) \leftarrow \text{BackBatchNorm}(g_{a_k}, s_k, \theta_k)$

$g_{a_{k-1}}^b \leftarrow g_{s_k} W_k^b$

$g_{W_k^b} \leftarrow g_{s_k}^\top a_{k-1}^b$

end for

While updating the weights, the following is done:

Each real valued weight, w^r , is constrained to remain between -1 and +1. If a weight update brings w^r outside $[-1, 1]$, it is clipped. This is done because otherwise, the real-valued weights will grow very large without having any impact on the binary weights, w^b . The new updated binary weights are then calculated as $w^b = \text{Sign}(w^r)$.

This network has the following layers:



- Fully connected (128)
- Ramp - rectified linear unit (ReLU) activation function
- Binarize activations
- Fully connected (128)
- Ramp - ReLU activation function
- Binarize activations
- Fully connected (10)
- Sigmoid activation function
- <https://blog.csdn.net/stdcoutzyx/article/details/50926174>

- <https://duanyzhi.github.io/Binary-Network/>
- BinaryConnect: Training Deep Neural Networks with binary weights during propagations
- Neural Networks with Few Multiplications
- Binary Neural Networks
- Code: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1
- Code Sample: Optimizing Binarized Neural Networks on Intel® Xeon® Scalable Processors
- Accelerating Neural Networks with Binary Arithmetic
- The High-Dimensional Geometry of Binary Neural Networks
- Binarized Neural Networks: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1
- Boolean Circuits are Neural Networks
- FP-BNN: Binarized neural network on FPGA
- Embedded Binarized Neural Networks
- Deep Learning for Real-Time Crime Forecasting and Its Ternarization
- <https://www.ibm.com/blogs/research/2019/04/2-bit-precision/>

Ternary Weight Networks

Ternary weight networks (TWNs) is the neural networks with weights constrained to +1, 0 and -1. This network can date up to the paper [Learning algorithms with neural network with ternary weights](#) in 1988.

- [Learning algorithms with neural network with ternary weights](#)
- [Ternary Weight Networks](#)
- [TBN: Convolutional Neural Network with Ternary Inputs and Binary Weights](#)
- [Trained Ternary Quantization](#)
- <https://iceory.github.io/2018/04/04/ternary-weight-networks/>
- <https://ieeexplore.ieee.org/document/8581485>

XOR-Net

In Binary-WeightNetworks, the filters are approximated with binary values resulting in 32× memory saving. In XNOR-Networks, both the filters and the input to convolutional layers are binary. [XNOR-Networks](#) approximate convolutions using primarily binary operations. This results in 58× faster convolutional operations and 32× memory savings.

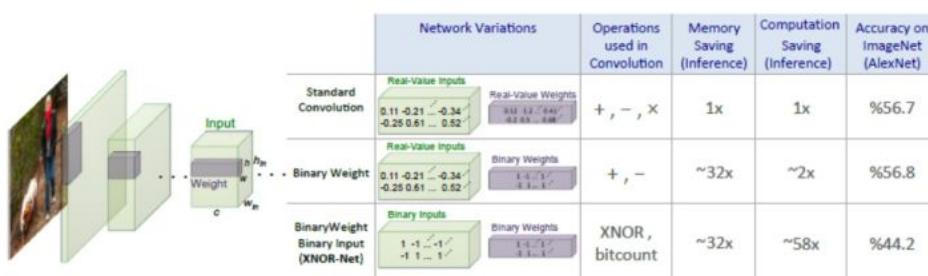


Fig. 1: We propose two efficient variations of convolutional neural networks. **Binary-Weight-Networks**, when the weight filters contains binary values. **XNOR-Networks**, when both weigh and input have binary values. These networks are very efficient in terms of memory and computation, while being very accurate in natural image classification. This offers the possibility of using accurate vision techniques in portable devices with limited resources.

- [XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks](#)

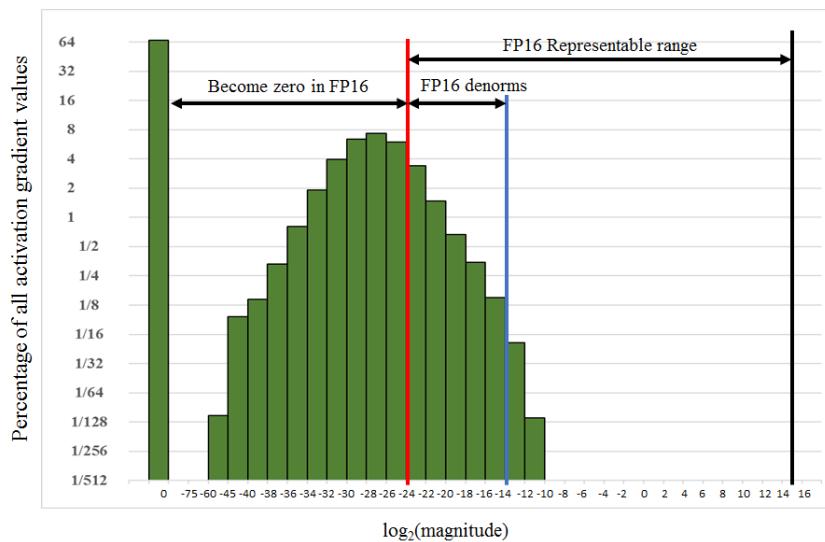
- [GNNOR-Net: Training deep neural networks with ternary weights and activations without full-precision memory under a unified discretization framework](#)
 - [XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks](#)
 - Joseph Chet Redmon: author of XNORNet
 - [XNOR-Net论文解读](#)
 - <https://github.com/liuzechun/Bi-Real-net>
 - [http://iis-projects.ee.ethz.ch/index.php/Bringing_XNOR-nets_\(ConvNets\)_to_Silicon](http://iis-projects.ee.ethz.ch/index.php/Bringing_XNOR-nets_(ConvNets)_to_Silicon)
 - http://web.eng.tau.ac.il/deep_learn/wp-content/uploads/2017/03/Binary-Deep-Learning.pdf
 - <https://papers.nips.cc/paper/6638-towards-accurate-binary-convolutional-neural-network.pdf>
 - <https://arxiv.org/pdf/2001.05936v1.pdf>
-

- [Low Precision Arithmetic Simulation in PyTorch](#)
- [Deep Learning with Limited Numerical Precision](#)
- [Making Neural Nets Work With Low Precision](#)
- [8-bit Inference with TensorRT](#)

Mixed Precision Training

Mixed-precision training lowers the required resources by using lower-precision arithmetic, which has the following benefits.

- Decrease the required amount of memory. Half-precision floating point format (FP16) uses 16 bits, compared to 32 bits for single precision (FP32). Lowering the required memory enables training of larger models or training with larger minibatches.
- Shorten the training or inference time. Execution time can be sensitive to memory or arithmetic bandwidth. Half-precision halves the number of bytes accessed, thus reducing the time spent in memory-limited layers.



The Y-axis is the percentage of all values on a log scale. The X-axis is the log scale of absolute values, as well as a special entry for zeros. For example, in this training session 66.8% of values were zero, whereas 4% of values were between 2^{-32} and 2^{-30} . A very efficient way to ensure that gradients fall into the range representable by half precision is to multiply the training loss with the scale factor. This adds just a single multiplication and by the chain rule it ensures that all the gradients are scaled up (or shifted up) at no additional cost. Loss scaling ensures that relevant gradient values lost to zeros are recovered. Weight gradients need to be scaled down

by the same factor S before the weight update. The scale-down operation could be fused with the weight update itself (resulting in no extra memory accesses) or carried out separately.

Additions to the traditional iteration procedure are in bold.

1. Make an FP16 copy of the weights

2. Forward propagate using FP16 weights and activations

3. Multiply the resulting loss by the scale factor S

4. Backward propagate using FP16 weights, activations, and their gradients

5. Multiply the weight gradients by 1/S

6. Optionally process the weight gradients (gradient clipping, weight decay, etc.)

7. Update the master copy of weights in FP32

- https://nnabla.readthedocs.io/en/latest/python/tutorial/mixed_precision_training.html
- <https://nvidia.github.io/OpenSeq2Seq/html/mixed-precision.html>
- <https://pdc.one/2019/05/14/Mixed-precision-training/>
- Paulius Micikevicius's talk "Training Neural Networks with Mixed Precision: Theory and Practice" (GTC 2018, S8923).
- Experimental Evaluation of Mixed Precision Training for End to End Applications
- Apex (A PyTorch Extension)
- <https://blog.masterliu.net/tensorflow/tensorflow-mixed-precision/>
- Mixed-Precision Training of Deep Neural Networks
- Mixed Precision Training
- User Guide
- TRAINING WITH MIXED PRECISION
- NVIDIA Apex: Tools for Easy Mixed-Precision Training in PyTorch
- <https://developer.nvidia.com/vulkan-driver>

Blended Coarse Gradient Descent

Coarse gradient is generally not a gradient of any function but an artificial ascent direction. The weight update of BCGD goes by coarse gradient correction of a weighted average of the full-precision weights and their quantization (the so-called blending), which yields sufficient descent in the objective value and thus accelerates the training.

- <https://www.math.uci.edu/~jxin/>
- Blended coarse gradient descent for full quantization of deep neural networks
- Training Quantized Deep Neural Networks and Applications with Blended Coarse Gradient Descent
- Blended Coarse Gradient Descent for Full Quantization of Deep Neural Networks
- <https://dominikfhg.github.io/TensorQuant/>
- Quantized Neural Networks: Training Neural Networks with Low Precision Weights and Activations
- Median Binary-Connect Method and a Binary Convolutional Neural Network for Word Recognition
- BinaryConnect: Training Deep Neural Networks with binary weights during propagations
- Binary Relax: A Relaxation Approach For Training Deep Neural Networks With Quantized Weights
- Quantization and Training of Low Bit-Width Convolutional Neural Networks for Object Detection

- <https://www.math.uci.edu/~jxin/signals.html>
- https://www.researchgate.net/scientific-contributions/43925792_Dimitris_S_Papailiopoulos

Low-precision Training

It turns out that DNNs can work with smaller datatypes, with less precision, such as 8-bit integers. Roughly speaking, we're trying to work with a number line looking closer to the sparse one on the bottom. The numbers are quantized, i.e. discretized to some specific values, which we can then represent using integers instead of floating-point numbers.

- <https://arxiv.org/abs/1906.03193>
- <https://arxiv.org/abs/1903.05662>
- <https://arxiv.org/pdf/1609.07061.pdf>

High-accuracy Low Precision

High-accuracy low precision (HALP) is our algorithm which runs SVRG and uses bit centering with a full gradient at every epoch to update the low-precision representation. It can do better than previous algorithms because it reduces the two sources of noise that limit the accuracy of low-precision SGD: gradient variance and round-off error.

- To reduce noise from gradient variance, HALP uses a known technique called **stochastic variance-reduced gradient** (SVRG). SVRG periodically uses full gradients to decrease the variance of the gradient samples used in SGD.
- To reduce noise from quantizing numbers into a low-precision representation, HALP uses a new technique we call **bit centering**. The intuition behind bit centering is that as we get closer to the optimum, the gradient gets smaller in magnitude and in some sense carries less information, so we should be able to compress it. By dynamically re-centering and re-scaling our low-precision numbers, we can lower the quantization noise as the algorithm converges.
- [High-Accuracy Low-Precision Training](#)
- [Training deep neural networks with low precision multiplications](#)
- [HALP: High-Accuracy Low-Precision Training](#)

Ultra-Low Precision Training

There are three primary challenges that make it difficult to scale precision below 16 bits while fully preserving model accuracy. Firstly, when all the operands (i.e., weights, activations, errors, and gradients) for general matrix multiplication (GEMM) and convolution computations are simply reduced to 8 bits, most DNNs suffer noticeable accuracy degradation. Secondly, reducing the bit precision of accumulations in GEMM from 32 bits to 16 bits significantly impacts the convergence of DNN training. This is why commercially available hardware platforms exploiting scaled precision for training (including GPUs) still continue to use 32 bits of precision for accumulation. Reducing accumulation bit precision below 32 bits is critically important for reducing the area and power of 8-bit hardware. Finally, reducing the bit precision of weight updates to 16-bit floating-point impacts accuracy, [while 32-bit weight updates, used in today's systems, require an extra copy of the high-precision weights and gradients to be kept in memory, which is expensive](#).

- [Accumulation Bit-Width Scaling For Ultra-Low Precision Training Of Deep Networks](#)
- [Ultra-Low-Precision Training of Deep Neural Networks](#)
- [8-Bit Precision for Training Deep Learning Systems](#)
- [Training High-Performance and Large-Scale Deep Neural Networks with Full 8-bit Integers](#)

- Highly Accurate Deep Learning Inference with 2-bit Precision
- OptQuant: Distributed training of neural networks with optimized quantization mechanisms
- <https://www.ibm.com/blogs/research/author/chia-yuchen/>
- <https://www.ibm.com/blogs/research/author/naigangwang/>

ADMM

We can apply alteranting direction method of mulipliers(ADMM) to train deep neural networks. The first part of ADMM-NN is a systematic, joint framework of DNN weight pruning and quantization using ADMM. It can be understood as a smart regularization technique with regularization target dynamically updated in each ADMM iteration, thereby resulting in higher performance in model compression than prior work. The second part is hardware-aware DNN optimizations to facilitate hardware-level implementations. Without accuracy loss, we can achieve 85\times and 24\times pruning on LeNet-5 and AlexNet models, respectively, significantly higher than prior work. The improvement becomes more significant when focusing on computation reductions. Combining weight pruning and quantization, we achieve 1,910\times and 231\times reductions in overall model size on these two benchmarks, when focusing on data storage. Highly promising results are also observed on other representative DNNs such as VGGNet and ResNet-50.

- <https://web.northeastern.edu/yanzhiwang/publications/>
- Deep ADMM-Net for Compressive Sensing MRI
- Extremely Low Bit Neural Network: Squeeze the Last Bit Out with ADMM
- Toward Extremely Low Bit and Lossless Accuracy in DNNs with Progressive ADMM
- ADMM-based Weight Pruning for Real-Time Deep Learning Acceleration on Mobile Devices
- ADMM-NN: An Algorithm-Hardware Co-Design Framework of DNNs Using Alternating Direction Method of Multipliers
- StructADMM: A Systematic, High-Efficiency Framework of Structured Weight Pruning for DNNs
- <https://blog.csdn.net/XSYYMY/article/details/81904882>
- <https://arxiv.org/abs/1706.06197>
- <https://csyhhu.github.io/>
- <https://www.ntu.edu.sg/home/sinnopan/>
- <https://ywang393.expressions.syr.edu/>
- <https://cse.buffalo.edu/~wenyaoxu/>
- <http://gr.xjtu.edu.cn/web/jiansun/publications>

Huffman Encoding

Huffman code is a type of optimal prefix code that is commonly used for loss-less data compression. It produces a variable-length code table for encoding source symbol. The table is derived from the occurrence probability for each symbol. As in other entropy encoding methods, more common symbols are represented with fewer bits than less common symbols, thus save the total space.

- Huffman coding is a code scheme.

Knowledge Distillation

Distillation (Hinton et al., 2015) is a kind of model compression approaches in which a pre-trained large model teaches a smaller model to achieve the similar prediction performance. It is often named as the "teacher-student" training, where the large model is the teacher and the smaller model is the student.

Initially, it is used to compress the knowledge in an ensemble into a single model which is much easier to deploy.

The core idea is that an obvious way to transfer the generalization ability of the cumbersome model to a small model is to use the class probabilities produced by the cumbersome model as "soft targets" for training the small model. When the soft targets have high entropy, they provide much more information per training case than hard targets and much less variance in the gradient between training cases, so the small model can often be trained on much less data than the original cumbersome model and using a much higher learning rate.

With distillation, knowledge can be transferred from the teacher model to the student by minimizing a loss function to recover the distribution of class probabilities predicted by the teacher model. In most situations, the probability of the correct class predicted by the teacher model is very high, and probabilities of other classes are close to 0, which may not be able to provide extra information beyond ground-truth labels. To overcome this issue, a commonly-used solution is to raise the temperature of the final softmax function until the cumbersome model produces a suitably soft set of targets. The soften probability q_i of class i is calculated from the logit z_i

$$q_i = \frac{\exp(z_i/T)}{\sum_j \exp(z_j/T)}$$

where T is the temperature. As T grows, the probability distribution is more smooth, providing more information as to which classes the cumbersome model more similar to the predicted class. It is better to include the standard loss ($T = 1$) between the predicted class probabilities and ground-truth labels. The overall loss function is given by:

$$L(x; W) = H(y, \sigma(z_s; T = 1)) + \alpha \cdot H(\sigma(z_t; T = \tau), \sigma(z_s, T = \tau))$$

where x is the input, W are parameters of the distilled small model and y is ground-truth labels, σ is the softmax parameterized by temperature T , H is the cross-entropy loss, and α is the coefficient of distillation loss.

- https://nervanasystems.github.io/distiller/knowledge_distillation/index.html
- Awesome Knowledge Distillation
- knowledge distillation papers
- <https://pocketflow.github.io/distillation/>
- Distilling the Knowledge in a Neural Network

Knowledge Distillation (KD) is a widely used approach to transfer the output information from a heavy network to a smaller network for achieving higher performance. The student network can be optimized using the following loss function based on knowledge distillation:

$$\mathcal{L}_{KD} = \frac{1}{n} H(y_S^i, y_T^i).$$

Therefore, utilizing the knowledge transfer technique, a portable network can be optimized without the specific architecture of the given network.

Algorithm 1 DFL for learning portable student networks.

Input: A given teacher network \mathcal{N}_T , parameters of different objects: α and β .

- 1: Initialize the generator G , the student network \mathcal{N}_S with fewer memory usage and computational complexity;
- 2: **repeat**
- 3: **Module 1: Training the Generator.**
- 4: Randomly generate a batch of vector: $\{\mathbf{z}^i\}_{i=1}^n$;
- 5: Generate the training samples: $x \leftarrow G(\mathbf{z})$;
- 6: Employ the teacher network on the mini-batch:
 $[y_T, t, f_T] \leftarrow \mathcal{N}_T(x)$;
- 7: Calculate the loss function \mathcal{L}_{Total} (Fcn.7):
- 8: Update weights in G using back-propagation;
- 9: **Module 2: Training the student network.**
- 10: Randomly generate a batch of vector $\{\mathbf{z}^i\}_{i=1}^n$;
- 11: Utilize the generator on the mini-batch: $x \leftarrow G(\mathbf{z})$;
- 12: Employ the teacher network and the student network on the mini-batch simultaneously:
 $y_S \leftarrow \mathcal{N}_S(x), y_T \leftarrow \mathcal{N}_T(x)$;
- 13: Calculate the knowledge distillation loss:
 $\mathcal{L}_{KD} \leftarrow \frac{1}{n} \sum_i \mathcal{H}(y_S^i, y_T^i)$;
- 14: Update weights in \mathcal{N}_S according to the gradient;
- 15: **until** convergence

Output: The student network \mathcal{N}_S .

- Great Breakthrough! Huawei Noah's Ark Lab first pioneers a novel knowledge distillation technique without training data.
- DAFL: Data-Free Learning of Student Networks
- <https://github.com/huawei-noah/DAFL>
- <https://blog.csdn.net/xbinworld/article/details/83063726>

Transferred/Compact Convolutional Filters

Transfer learning methods have demonstrated state-of-the-art performance on various small-scale image classification tasks. This is generally achieved by exploiting the information from an ImageNet convolution neural network (ImageNet CNN). However, the transferred CNN model is generally with high computational complexity and storage requirement. It raises the issue for real-world applications, especially for some portable devices like phones and tablets without high-performance GPUs. Several approximation methods have been proposed to reduce the complexity by reconstructing the linear or non-linear filters (responses) in convolutional layers with a series of small ones.

- EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks
- MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications
- Compact Convolutional Neural Network Transfer Learning For Small-Scale Image Classification
- <https://zhuanlan.zhihu.com/p/35405071>
- <https://www.jianshu.com/u/4a95cf020626>
- <https://github.com/MG2033/ShuffleNet>

Tensor Methods

Note that the deep learning models are composite of linear and non-linear maps. And linear maps are based on matrices.

These methods take a layer and decompose it into several smaller layers. Although there will be more layers after the decomposition, the total number of floating point operations and weights will be smaller. Some reported results are on the order of x8 for entire networks (not aimed at large tasks like imangenet, though), or x4 for specific layers inside imangenet. *My experience was that with these decompositions I was able to get a speedup of between x2 to x4, depending on the accuracy drop I was willing to take.*

- <http://www.tensorlet.com/>
- Optimal Low Rank Tensor Factorization for Deep Learning
- <https://github.com/jacobgil/pytorch-tensor-decompositions>
- Accelerating deep neural networks with tensor decompositions
- Workshop: "Deep Learning and Tensor/Matrix Decomposition for Applications in Neuroscience", November 17th, Singapore
- <http://www.deeptensor.ml/index.html>
- <http://tensorly.org/stable/home.html>
- <http://www.vision.jhu.edu/>
- https://wabyking.github.io/talks/DL_tensor.pdf

Singular value decomposition

The matrix $A_{m \times n}$ can be decomposed as the multiplication of two matrices such as $A_{m \times n} = Q_{m \times r} R_{r \times n}$, so that the storage is from $O(m \times n)$ to $O(m + n) \times O(r)$.

To explore a low-rank subspace combined with a sparse structure for the weight matrix W , we assume that $W \approx L + S$, where L is a low-rank component and S is a sparse matrix. Then, to compress the weight matrix, we have the following model:

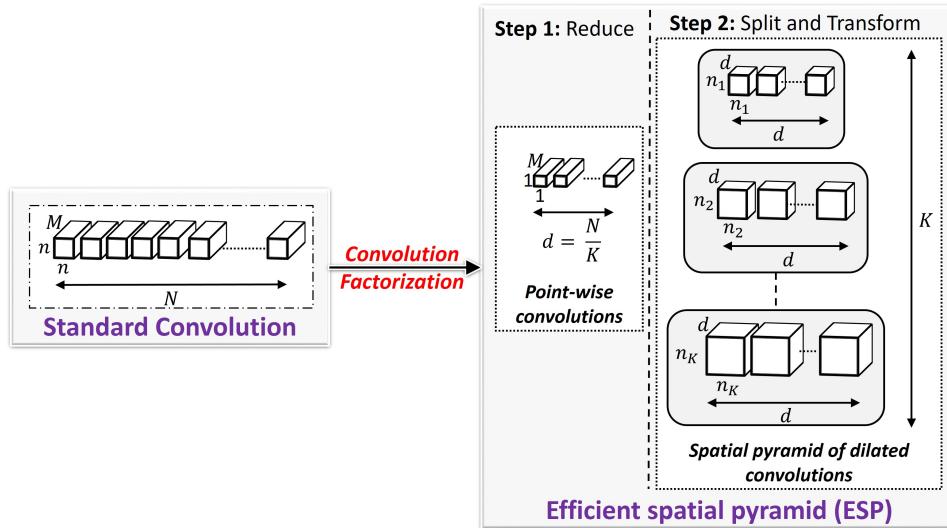
$$\begin{aligned} & \min_{L,S} \frac{1}{2} \|W - L - S\|_F^2, \\ & \text{s.t. } \text{rank}(L) \leq r, \\ & \quad \text{card}(S) \leq c, \end{aligned}$$

where $\text{rank}(L)$ denotes the rank of L and $\text{card}(S)$ denotes the cardinality of matrix S .

And Toeplitz Matrix can be applied to approximate the weight matrix

$$W = \alpha_1 T_1 T_2^{-1} + \alpha_2 T_3 T_4^{-1} T_5$$

where M is the square weight matrix, T_1, T_2, T_3, T_4, T_5 are square *Toeplitz matrix*.



- https://en.wikipedia.org/wiki/Low-rank_approximation
- Low Rank Matrix Approximation
- On Compressing Deep Models by Low Rank and Sparse Decomposition
- ESPNets for Computer Vision Efficient CNNs for Edge Devices
- <https://www.cnblogs.com/zhonghuasong/p/7821170.html>
- <https://github.com/chester256/Model-Compression-Papers>
- <https://arxiv.org/pdf/1712.01887.pdf>

-
- <https://srdas.github.io/DLBook/intro.html#effective>
 - <https://cognitiveclass.ai/courses/accelerating-deep-learning-gpu/>
 - <https://github.com/songhan/Deep-Compression-AlexNet>
 - awesome-model-compression-and-acceleration
 - gab41.lab41.org
 - CS 598 LAZ: Cutting-Edge Trends in Deep Learning and Recognition
 - http://slazebni.cs.illinois.edu/spring17/lec06_compression.pdf
 - http://slazebni.cs.illinois.edu/spring17/reading_lists.html#lec06
 - CS236605: Deep Learning
 - <https://mlperf.org/>

Privacy and Security

Existing security mechanisms for high-performance and distributed computing infrastructure are complex and difficult to deploy. As a result, many high-performance and distributed computing facilities do not deploy sufficient security mechanisms. This has prevented privacy-sensitive applications, such as those in the medical fields, and security-sensitive applications from using such facilities. In this project, we will develop and deploy DICE, Data Insurance in the Cluster Environment, to enhance the security in HPC and distributed computing clusters. DICE will consist of three major components: a container-based virtual cluster, a component to defend against side-channel attacks, and a secure execution ledger for auditing. The container-based virtual cluster will be developed based on the Docker Linux container. The Docker security mechanism will be enhanced by deploying an effective key management scheme for groups and by reducing the attack surface exposed to containers. Novel defense mechanisms will be developed and deployed to defend against side-channel attacks in the cluster environment by exploiting new security features in the recent processors. The secure execution ledger will provide a global holistic view of program execution in the whole system, enabling auditing the behavior of individual user as well as user groups. DICE essentially creates a two-level security

model: on the (physical) cluster level, a group of (mostly) mutually trusted users share a single virtual cluster for their jobs; and inside the virtual cluster, the group may use existing security mechanisms of their software-of-choice to further refine security.



- Deep Leakage from Gradients
- <http://www.cs.cornell.edu/~shmat/research.html>

Distributed Training

The problem of deep learning $T(x; \Theta)$ is the big model and big data, i.e., Θ may be in too extra-high dimensional space to store in a single laptop computer. And the training process is to find the optimal parameters $\arg \min_{\Theta} \sum_i L(T(x_i; \Theta), y_i)$, which requires sufficient data size.

Sometimes we need partition the model or the data into different machines. In another world, the model or the data are distributed in a few machines.

Distributed training of deep learning models is a branch of distributed computation.

Training advanced deep learning models is challenging. Beyond model design, model scientists also need to set up the state-of-the-art training techniques such as distributed training, mixed precision, gradient accumulation, and checkpointing. Yet still, scientists may not achieve the desired system performance and convergence rate. Large model sizes are even more challenging: a large model easily runs out of memory with pure data parallelism and it is difficult to use model parallelism.

It is really important to reduce the cost of communication in distributed computation including the communication time, communication frequency, communication content and latency.

- BigDL is a distributed deep learning library for Apache Spark
- <https://github.com/intel-analytics/BigDL/>
- <https://github.com/amplab/drizzle-spark>
- <https://github.com/NVIDIA/ncc>
- <https://github.com/horovod/horovod>
- <http://maxpumperla.com/elephas/>
- <https://horovod.ai/>
- <http://explorer.cs.fsu.edu/>
- https://www.cs.rice.edu/~as143/COMP640_Fall16/
- <https://www.cs.rice.edu/~as143/>
- <https://www.deepspeed.ai/>
- <https://www.comp.hkbu.edu.hk/~chxw/>

Accelerating Deep Learning Workloads

- <https://itpeernetwork.intel.com/accelerating-deep-learning-workloads/#gs.ny4nke>
- <https://github.com/bytedance/bytdeps>
- A Generic Communication Scheduler for Distributed DNN Training Acceleration
- Characterizing Deep Learning Training Workloads on Alibaba-PAI
- <https://arxiv.org/pdf/1806.03377.pdf>
- <http://www.iiswc.org/iiswc2019/index.html>
- <https://cs.stanford.edu/~matei/>
- <https://www.microsoft.com/en-us/research/project/fiddle/>

- <https://github.com/msr-fiddle/pipedream>
- <https://github.com/msr-fiddle/philly-traces>
- PipeMare: Asynchronous Pipeline Parallel DNN Training
- GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism
- XPipe: Efficient Pipeline Model Parallelism for Multi-GPU DNN Training
- <https://arxiv.org/pdf/1806.03377.pdf>
- <https://www.microsoft.com/en-us/research/project/fiddle/>
- Bandwidth Optimal All-reduce Algorithms for Clusters of Workstations
- <https://zhuanlan.zhihu.com/p/87515411>
- <https://zhuanlan.zhihu.com/p/84862107>
- Accelerating Deep Learning Workloads through Efficient Multi-Model Execution
- <https://cs.stanford.edu/~matei/>

PipeDream

PipeDream, a system developed as part of Microsoft Research's Project Fiddle, introduces pipeline parallelism, a new way to parallelize DNN training by combining traditional intra-batch parallelism (model and data parallelism) with inter-batch parallelism (pipelining).

- PipeDream: Generalized Pipeline Parallelism for DNN Training

Count-Sketches

Sketch is a class of algorithms using a probabilistic data structure to approximate the distribution of input data.

The count-sketches streaming algorithm instantiates the following framework:

1. Find a randomized streaming algorithm whose output (as a random variable) has the desired expectation but usually high variance (i.e., noise).
 2. To reduce the variance/noise, run many independent copies in parallel and combine their outputs.
- Compressing Gradient Optimizers via Count-Sketches
 - MISSION: Ultra Large-Scale Feature Selection using Count-Sketches
 - <https://arxiv.org/abs/1804.10673>
 - <https://github.com/rdspring1/Count-Sketch-Optimizers>
 - <http://net.pku.edu.cn/~yangtong/>
 - <https://papers.nips.cc/paper/9168-memory-efficient-adaptive-optimization.pdf>
 - <https://dblp.org/pers/hd/k/Kyriakidis>Anastasios>
 - <https://www.csauthors.net/vijai-mohan/>
 - SketchML: Accelerating Distributed Machine Learning with Data Sketches

Efficient Communication

- <https://www.usenix.org/conference/hotedge18/presentation/tao>
- eSGD: Communication Efficient Distributed Deep Learning on the Edge
- <https://www.run.ai/>
- <https://theaisummer.com/distributed-training/>
- Communication-Efficient Learning of Deep Networks from Decentralized Data
- <https://www.usenix.org/system/files/conference/atc17/atc17-zhang.pdf>
- <https://embedl.ai/>

- Communication-Efficient Federated Deep Learning With Layerwise Asynchronous Model Update and Temporally Weighted Aggregation
- Optimus: An Efficient Dynamic Resource Scheduler for Deep Learning Clusters
- CS-345, Spring 2019: Advanced Distributed and Networked Systems

DeepSpeed

The DeepSpeed API is a lightweight wrapper on PyTorch. This means that you can use everything you love in PyTorch and without learning a new platform. In addition, DeepSpeed manages all of the boilerplate state-of-the-art training techniques, such as distributed training, mixed precision, gradient accumulation, and checkpoints so that you can focus on your model development. Most importantly, you can leverage the distinctive efficiency and effectiveness benefit of DeepSpeed to boost speed and scale with just a few lines of code changes to your PyTorch models.

- <https://www.microsoft.com/en-us/research/blog/zero-2-deepspeed-shattering-barriers-of-deep-learning-speed-scale/>
- <https://www.deepspeed.ai/>

Gradient Code and Compression

Gradient code and compression is to accelerate the distributed training of deep learning models by reducing the communication cost.

Gradient Code and Approximate Gradient Coding

Some nodes may be much slower than others parallelized or distributed computation environment. With redundancy, we may not need every node to finish. See [prior articles](#) for more articles on this topic.

Approximate gradient coding allows us to tolerate more stragglers with less work.

- Gradient Coding
- Reducing the Average Delay in Gradient Coding
- Tree Gradient Coding
- Communication-computation efficient gradient coding
- Approximate Gradient Coding via Sparse Random Graphs
- ErasureHead: Distributed Gradient Descent without Delays Using Approximate Gradient Coding
- https://users.oden.utexas.edu/~leiqi/stragglers_nips.pdf
- https://zachcharles.com/files/Presentations/sbc_slides.pdf
- Ternary Gradients to Reduce Communication in Distributed Deep Learning (TensorFlow)

- <https://hwang595.github.io/publications/>
- <http://papail.io/>
- <http://kangwooklee.com/>
- <https://www.ntu.edu.sg/home/hmkiah/>
- <https://users.oden.utexas.edu/~leiqi/>
- <https://zachcharles.com/>

Gradient Compression

In distributed training of machine learning models with stochastic optimization, the exchange of parameter updates between workers often is a bottleneck that limits the scalability

of distributed training. This is especially true for models with a large parameter space, such as neural networks. Several techniques have been proposed to enhance scalability by compressing gradients, e.g. by sending a sparse set of coordinates only, or by quantization. We study the gradient compression literature from both sides: on the one hand, we study properties of these algorithms in a distributed setting, and their effectiveness for speed and scalability. On the other hand, we explore properties of the minima found by these algorithms, such as robustness or generalisation.

Deep Gradient Compression @ MIT

- Deep Gradient Compression: Reducing the Communication Bandwidth for Distributed Training
- <https://blog.csdn.net/a609640147/article/details/90054754>
- <https://songhan.mit.edu/>
- Deep Leakage from Gradients
- <https://littleorange.site/2019/02/deep-gradient-compression.html>
- Training Kinetics in 15 Minutes: Large-scale Distributed Training on Videos

Gradient Compression @ epfl

We study gradient compression methods to alleviate the communication bottleneck in data-parallel distributed optimization. Despite the significant attention received, current compression schemes either do not scale well or fail to achieve the target test accuracy. We propose a new low-rank gradient compressor based on power iteration that can i) compress gradients rapidly, ii) efficiently aggregate the compressed gradients using all-reduce, and iii) achieve test performance on par with SGD. The proposed algorithm is the only method evaluated that achieves consistent wall-clock speedups when benchmarked against regular SGD with an optimized communication backend. We demonstrate reduced training times for convolutional networks as well as LSTMs on common datasets.

- Gradient Compression Techniques to Accelerate Distributed Training of Neural Networks
- QSGD: Communication-Efficient SGD via Gradient Quantization and Encoding
- Sparse Binary Compression: Towards Distributed Deep Learning with minimal Communication
- ATOMO: Communication-efficient Learning via Atomic Sparsification
- Error Feedback Fixes SignSGD and other Gradient Compression Schemes
- <https://github.com/epfml/error-feedback-SGD>
- PowerSGD: Practical Low-Rank Gradient Compression for Distributed Optimization
- <https://github.com/epfml/powersgd>
- Decentralized SGD and Consensus with Communication Compression

-
- <https://sstich.ch/>
 - <https://tvogels.nl/>
 - <https://parsa.epfl.ch/~falsafi/>
 - <https://lts4.epfl.ch/people/people-current/frossard-2/>
 - <http://www.martinjaggi.ch/>
 - <http://www.sysml.cc/>
 - <https://sstich.ch/>
 - <https://www.epfl.ch/labs/mlo/>

Gradient Compression @ Edinburgh

- Sparse Communication for Distributed Gradient Descent
- <https://www.kheafield.com/>
- <https://www.kheafield.com/papers/>
- <http://fusharblog.com/archive/>

Gradient Compression @ kaust

- Gradient compression for distributed training of machine learning models
- https://www.maths.ed.ac.uk/~prichtar/i_papers.html
- https://richtarik.org/i_papers.html
- https://richtarik.org/i_team.html
- https://www.maths.ed.ac.uk/~prichtar/i_seminar.html
- Better Communication Complexity for Local SGD
- Gradient Descent with Compressed Iterates

Synthetic gradient

It's a simple idea: rather than compute gradients through backpropagation, we can train a model to predict what those gradients will be, and use our prediction to update our weights. It's dynamic programming for neural networks.

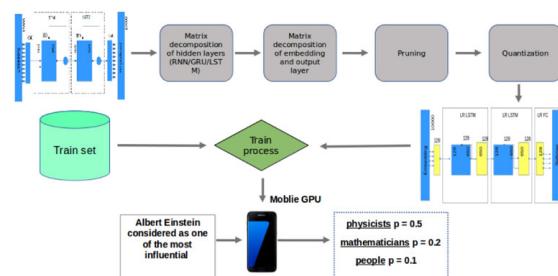
- <https://github.com/andrewliao11/dni.pytorch>
- <https://arxiv.org/pdf/1703.00522.pdf>
- <https://r2rt.com/synthetic-gradients-with-tensorflow.html>
- <https://arxiv.org/abs/1703.00522>
- <http://deliprao.com/archives/187>
- Exploring the Use of Synthetic Gradients for Distributed Deep Learning across Cloud and Edge Resources
- <http://cnichkawde.github.io/SyntheticGradients.html>

Gradient Centralization

- <http://www.breloff.com/>
- <https://github.com/Yonghongwei/Gradient-Centralization>
- <https://github.com/lessw2020/Ranger-Deep-Learning-Optimizer>
- https://www4.comp.polyu.edu.hk/~cslzhang/paper/conf/ECCV20/ECCV_GC.pdf

Compressing Recurrent Neural Network

All techniques above can be used to fully-connected networks or generally feed-forward network. RNN is feedback network where there is rings in its computational graph.



Proposed pipeline for the compression of recurrent neural nets in the language modeling problem. For such recurrent neural nets we firstly compress their internal recurrent layers, then we continue by compression of external embedding layers and the output one. In addition, we have shown how to apply pruning and quantization for neural nets compression. We implemented an optimized inference engine of compressed recurrent neural nets for a mobile phone with GPU

- Dynamically Hierarchy Revolution: DirNet for Compressing Recurrent Neural Network on Mobile Devices

- Compressing Recurrent Neural Network with Tensor Train
- Learning Compact Recurrent Neural Networks with Block-Term Tensor Decomposition
- ANTMAN: SPARSE LOW-RANK COMPRESSION TO ACCELERATE RNN INFERENCE
- Deep Comresion
- Run-Time Efficient RNN Compression for Inference on Edge Devices
- LSTM compression for language modeling
- Parameter Compression of Recurrent Neural Networks and Degradation of Short-term Memory
- <https://ai.google/research/pubs/pub44632>
- CLINK: Compact LSTM Inference Kernel for Energy Efficient Neurofeedback Devices

Compressing GANs

- <https://arxiv.org/abs/1902.00159>
- <https://github.com/amitadate/gan-compression>
- Adversarial Network Compression
- Model Compression with Generative Adversarial Networks
- <https://github.com/huawei-noah/GAN-pruning>
- https://github.com/kedartatwawadi/NN_compression
- Optimizing Data-Intensive Computations in Existing Libraries with Split Annotations

Products and Packages

- 开源背后 | 面对端侧推理引擎的挑战，阿里工程师如何应对？ - 阿里技术的文章 - 知乎
- <https://zhuanlan.zhihu.com/p/87392811>
- <https://zhuanlan.zhihu.com/p/87458316>
- <https://lutzroeder.github.io/netron/>
- <https://github.com/daquexian/onnx-simplifier>
- <https://www.dataiku.com/>

Edge Computation

Edge computation is to perform some computation on the edge devices such as the monior in order to send less raw data to the computation center.

Machine learning models for edge devices need to have a small footprint in terms of storage, prediction latency, and energy. One instance of where such models are desirable is resource-scarce devices and sensors in the Internet of Things (IoT) setting. Making real-time predictions locally on IoT devices without connecting to the cloud requires models that fit in a few kilobytes.

- <https://github.com/microsoft/EdgeML>
- <https://github.com/fritzlabs/Awesome-Mobile-Machine-Learning>
- <https://github.com/Bisonai/awesome-edge-machine-learning>
- <https://github.com/hoangcuong2011/Good-Papers>
- <https://github.com/rkirkendall/MLEdgeDeploy>
- <https://github.com/sig-ml/bleedml>
- <https://www.usenix.org/conference/hotedge18>
- <https://project-sparthan.github.io/>

Mobile deep learning is aimed to run deep learning models (training or inference) on the mobile phones.

It is necessary to compress the deep learning models in order to run it in mobile phones.

- <https://www.sigmobile.org/mobisys/2019/workshops/deepmobile19/>
 - <https://github.com/amusi/mobile-deep-learning>
 - <https://github.com/pytorch/qnnpack>
 - <https://mi.inf.ed.ac.uk/>
 - <https://www-users.cs.umn.edu/~fengqian/>
 - <http://homepages.inf.ed.ac.uk/ppatras/>
 - <https://www.cs.ucr.edu/~jiasi/>
 - <https://ix.cs.uoregon.edu/~jiao/>
 - <https://arxiv.org/abs/1904.09274>
 - <https://github.com/PaddlePaddle/Paddle-Lite>
 - <https://paddlepaddle.github.io/Paddle-Lite/>
 - <https://github.com/alibaba/MNN>
 - <https://www.tensorflow.org/lite>
 - <https://project-sparthan.github.io/papers/acm19.pdf>
 - <https://www.zhqiang.org/deep-learning-mobile/>
 - <https://bdtechtalks.com/2019/12/30/computer-vision-applications-deep-learning/>
 - http://openaccess.thecvf.com/content_ICCVW_2019/papers/CEFRL/Chen_MuffNet_Multi-Layer_Feature_Federation_for_Mobile_Deep_Learning_ICCVW_2019_paper.pdf
 - Deep Learning for Mobile Multimedia: A Survey
-

DNN Acceleration Framework

<https://hgpu.org/>

NVIDIA

<https://developer.nvidia.com/cudnn>

<http://nvdla.org/>

<https://docs.nvidia.com/cuda/>

<https://developer.nvidia.com/tensorrt>

cupy

intel

ideep

[Intel\(R\) Math Kernel Library for Deep Neural Networks \(Intel\(R\) MKL-DNN\)](#)

<https://github.com/intel/onnxruntime>

nGraph, PlaidML

<https://intel.github.io/mkl-dnn/>

Reference workloads for modern deep learning methods.

DNN Acceleration Framework

[Minerva: a fast and flexible tool for deep learning on multi-GPU.](#)

[SigDL -- Deep Learning for IoT Device and Edge Computing Embedded Targets](#)

[Menoh: fast DNN inference library with multiple programming language support](#)

[trillium](#)

<https://github.com/alibaba/MNN>

<https://github.com/sql-machine-learning/elasticdl>

<https://github.com/Tencent/TNN>

Model Compression Packages

[Distiller is an open-source Python package for neural network compression research.](#)

[PocketFlow](#)

[PocketFlow中的模型压缩算法](#)

[BNN](#)

[PERMDNN: Efficient Compressed DNN Architecture with Permuted Diagonal Matrices](#)

[knowledge-distillation-pytorch](#)

[keras_compressor](#)

[TensorFlow Lite, tensorflow-compression](#)

[TensorRT](#)

<https://github.com/Tencent/ncnn>

[Introduction to Intel® Deep Learning Deployment Toolkit](#)

Tool kits

- <https://www.wandb.com/>
- <https://github.com/fastnlp/fitlog>
- <https://optuna.org/>
- <https://preferred.jp/en/projects/optuna/>