Life is Powered by Machine Learning

Most desirable modern features/services are powered by DNNs

- Object Localization
- Translation
- Speech Recognition
- Gesture Interpretation
- Image Restoration
- Gaming AI
- Medical Diagnosis
- Malware Detection
# Why is Machine Learning Exploding?

<table>
<thead>
<tr>
<th>Application (Data)</th>
<th>Algorithm (Model)</th>
<th>Hardware (Computing)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Big data has resulted in huge training corpus</td>
<td>Theory behind ML and DNN has made leaps and bounds in recent years</td>
<td>Transistor scaling allows for massively parallelized tasks</td>
</tr>
<tr>
<td>Use this data to intelligently solve previously intractable problems</td>
<td>Can now efficiently deploy solutions to both the cloud and the edge</td>
<td>Emerging architectures promise huge gains in energy efficiency</td>
</tr>
</tbody>
</table>
DNN Acceleration on Conventional Platforms

Efficiency vs. Flexibility

Mismatch: Algorithm vs. Hardware

<table>
<thead>
<tr>
<th></th>
<th>Algorithm</th>
<th>Hardware</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model/Component Scale</td>
<td>Large</td>
<td>Small/Moderate</td>
</tr>
<tr>
<td>Reconfigurability</td>
<td>Easy</td>
<td>Hard</td>
</tr>
<tr>
<td>Accuracy vs. Power</td>
<td>Accuracy</td>
<td>Tradeoff</td>
</tr>
<tr>
<td>Training Implementation</td>
<td>Easy</td>
<td>Hard</td>
</tr>
<tr>
<td>Precision vs. Limited</td>
<td>Double (high) precision</td>
<td>Low precision (often a few bits)</td>
</tr>
<tr>
<td>Programmability</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Connectivity Realization</td>
<td>Easy</td>
<td>Hard</td>
</tr>
</tbody>
</table>
New Architectures and Systems

Google Tenser Processing Unit (TPU)
• A custom ASIC specifically for machine learning — and tailored for TensorFlow.
• 10X better-optimized performance/watt for machine learning


IBM TrueNorth
• 4,096 neurosynaptic cores
• 1 million neurons
• 256 million synapses
• A 65mW real-time neurosynaptic processor


ARM-SpiNNake
• Human Brain Project
• 18 ARM968 embedded processors
• DDR SDRAM

SpiNNaker, http://apt.cs.manchester.ac.uk/projects/SpiNNaker/project/

Intel Loihi
• 128 neuromorphic cores
• 3 Lakemont x86 cores (Quark)
• 130,000 artificial neurons
• 130 million synapses
Outline

- Introduction
- ReRAM-based Neuromorphic Chip Design
- Cross-Layer Codesign for Efficiency & Reliability
- Conclusion
Memristor – Rebirth of Neuromorphic Circuits

Memristor, or Metal-oxide Resistive Random Access Memory a.k.a. ReRAM

Synapse Network

Programmable resistor w/ analog states

Memristor Crossbar

Natural matrix operation

\[ y_1 = \sum x_i \cdot g_{il} \]

High density

[DAC12, M. Hu et al.]
Memristor (ReRAM) for Computation

Brain-State-in-a-Box (BSB) Recall & Training

\[ \Delta A = lr \ast (X - AX) \otimes X \]

\[ X(t+1) = S(\alpha \cdot A \cdot X(t) + \lambda \cdot X(t)) \]

\[ A = A + \Delta A \]

Input vector \( V(0) \)

Comparers detect the converge status.

Next iteration

Summing op-amps perform analog voltage signal addition/subtraction

We need two memristor arrays since memristor can only represent positive weights.

M. Hu, et al., DAC 2012, TNNLS 2014
The Spike-based Neuromorphic Design

Integrate-and-fire Circuit (IFC)

Ideal Computation Result

\[ n_{y,j}(t) \propto \int_{\tau=0}^{t} \sum_{i=0}^{N-1} g_{ij} V_{x,i}(\tau) d\tau \]

Real Computation Result

\[ n_{y,j} = \frac{t_m}{\alpha} \left( \sum_{i=0}^{n-1} g_{ij} v_i \right) + t_0 \]

Crossbar

IFC
Processing Engine with In-Situ Neuron Activation (ISNA)

Computing Mode:  
\[ \text{Nin} \rightarrow 256b \text{ Input Images} \]

Memory Mode:  
\[ \text{Data Memory address} \rightarrow 16b \]

Proposed RRAM spiking nvCIM PE

[Image of diagram]

[B. Yan, et al., VLSI Symp. 2019]
Spike Conversion

Tradeoff between large input current range and response speed

- 5.12 GOPS @ 8-bit precision
- 200ns latency @ 8-bit precision
- 43x area reduction compared to ADC design by K. Ohhata (JSSC 2019)

[1. Yan, et al., VLSI Symp. 2019]
In Situ Nonlinear Activation Function

• How to adjust nonlinear region shape?

Vref: Read voltage of BL
  • Vref ↑, BL current ↑
  • Proportional tuning

Vth: Threshold of capacitor charging/discharging
  • Vth ↓, Charging/discharging ↑
  • Distorted tuning

B. Yan, et al., VLSI Symp. 2019
Configurable Activation Precision

- Tradeoff between high-precision and high-performance
- Useful for system-level fine-grained optimization
Measured Neural Network Results

**Technology**: 0.15µm CMOS +HfO RRAM

- **Macro capacity**: 64K (256×256)
- **Clock frequency**: 50MHz
- **Energy efficiency**: 0.257pJ/Mac
- **Average power**: 1.52 mW


[B. Yan, et al., VLSI Symp. 2019]
Adaptable Threshold Spike-timing Neuromorphic Design

• Neuron dynamics under TTFS scheme

• How to map the excitatory and inhibitory synapses to the ReRAM crossbar array?
• How to leverage the event-driven property of SNNs to reduce energy consumption as most neurons are in the idle state?
Design Overview
Design Details

- **Power-gating of Post-neuron Module**
  - The CA dominates the overall power due to the power-consuming opamp within it.
  - In the TTFS-based SNNs, each neuron fires only a single spike during each inference time window.
  - Applying power-gating to the post-neuron module.

- **Timing threshold adjustment (TTA)**
  - Earlier spikes represent stronger activations, while later spikes contribute less to neuron’s potential accumulation in the next layer.
  - TTA scheme speeds up one iteration by allowing only a portion of neurons in each layer to propagate their spikes.

![Graph showing energy comparison with and without power-gating.](image-url)
Evaluation

- Accuracy analysis of MFTA
- Tradeoff in TTA
- Energy & Speed
Cros-Layer Co-Design for Efficiency & Reliability
Why Sparse DNN Models?

- Ever-growing size brought challenges to the deployment of the DNN models
- The parameters of DNNs are redundant
- We can reduce the computation and bandwidth requirement by eliminating those redundancy

Sparse DNNs

- **Weight Sparsity**
  - Unstructured Pruning
  - Structured Pruning
- **Activation Sparsity**
- **Bit-level Sparsity**
- **Inherent Sparsity**
The Need of Structural Sparsity

- Non-structured sparsity may not bring much speedup on traditional platforms like GPUs.
- Structured sparsity is more hardware-friendly.
- Structured sparsity can be achieved by having all the parameters within a structured group become zero or nonzero simultaneously.

[NeurIPS’16, W. Wen et. al.]
Structurally Sparse Deep Neural Networks

In a nutshell, a group can be **any** form of a weight block, depending on what sparse structure you want to learn.

In CNNs, a group of weights can be a channel, a 3D filter, a 2D filter, a filter shape fiber (i.e., a weight column), and even a layer (in ResNets).
Group Lasso Regularization is **ALL** You Need

Step 1: Weights are split to $G$ groups $w^{(1..G)}$

\[ e.g. \ (w_1, w_2, w_3, w_4, w_5) \rightarrow \text{group 1: (}w_1, w_2, w_3\text{), group 2: (}w_4, w_5\text{)} \]

Step 2: Add group Lasso on each group $w^{(g)}$

\[ \|w^{(g)}\|_g = \sqrt{\sum_{i=1}^{G} |w^{(g)}_i|^2} \]

i.e. vector length

\[ \sqrt{w_1^2 + w_2^2 + w_3^2} + \sqrt{w_4^2 + w_5^2} \]

Step 3: Sum group Lasso over all groups as a regularization:

\[ R_g(w) = \sum_{g=1}^{G} \|w^{(g)}\|_g \]

\[ \sqrt{w_1^2 + w_2^2 + w_3^2} + \sqrt{w_4^2 + w_5^2} \]

Step 4: SGD optimizing

\[ \arg\min_w \left\{ E\left( w \right) \right\} = \arg\min_w \left\{ E_d\left( w \right) + \lambda \cdot R_g\left( w \right) \right\} \]

We refer to our method as **Structured Sparsity Learning (SSL)**
NIPS’16; ICLR’18; ICASSP’19
• is supported by the library of Intel Nervana Neural Network Processors.
• is adopted by Intel’s newest NLP accelerator.
• applied to Microsoft Bing
• is adopted by SF-Technology, achieving 2X performance improvement in their datacenter.

Structurally Sparse \textit{LeNet} – Removing Channels and Filters

Structurally Sparse Matrices

Structurally Sparse LSTMs: Removing Hidden Structures

Experiments

Removing layers in ResNets

K. He et al. CVPR 2016.

[ICLR’18 W. Wen et. al.], [ICASSP’19 J. Zhang et.al.]
Sparsity-inducing Regularizer for DNN

- **L1 regularizer (sum of absolute values)**
  - Used for sparsity since 1996
  - Differentiable, convex, easy to optimize
  - Proportional to the scaling, can only “scale down” all the elements with the same speed, undesired penalty on large elements

- **L0 regularizer (number of nonzero values)**
  - Reflect the sparsity by definition
  - Scale-invariant
  - No useful gradients
  - Need additional tricks for applying on DNN pruning (i.e. stochastic approximation, ADMM), making the problem complicated

Moving Beyond Lasso

- **Goal**: Find a sparsity-inducing regularizer that is both differentiable and scale-invariant

- **Hoyer-square regularizer**
  - Square of the L1/L2 ratio, differentiable, scale-invariant, same range and similar minima structure as L0

- **Group-HS regularizer for structural pruning**
  - Apply Hoyer-Square regularizer over the L2 norm of the group
  - Group weights along channels and filters

\[
G_H(W) = \frac{\left(\sum_{g=1}^{G} ||w^{(g)}||_2\right)^2}{\sum_{g=1}^{G} ||w^{(g)}||_2^2} = \frac{\left(\sum_{g=1}^{G} ||w^{(g)}||_2\right)^2}{||W||_2^2}
\]

[ICLR'20 H. Yang et.al.]
Optimizing with Hoyer-Square Regularizer

• Gradient

\[
\partial w_j H_S(W) = 2 \text{sign}(w_j) \frac{\sum_i |w_i|}{(\sum_i w_i^2)^2} \left( \sum_i w_i^2 - |w_j| \sum_i |w_i| \right).
\]

• Induced auto-trimming
  – Turn weights with smaller absolute value to zero while protecting larger weights
  – Gradually extend trimming threshold as more weights coming close to zero
High Element-wise Sparsity & Large FLOPs Reduction

• Further improve the model compression rate of element-wise pruning w/o accuracy loss

• Outperform the Pareto frontier of performance-#FLOPs tradeoff

Element-wise pruning results on AlexNet w/o accuracy loss

<table>
<thead>
<tr>
<th>Layer</th>
<th>Baseline</th>
<th>Han et al.</th>
<th>Zhang et al.</th>
<th>Ma et al.</th>
<th>Hoyer</th>
<th>HS</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONV1</td>
<td>34.8K</td>
<td>29.3K</td>
<td>28.2K</td>
<td>24.2K</td>
<td>21.3K</td>
<td>31.6K</td>
</tr>
<tr>
<td>CONV2</td>
<td>307.2K</td>
<td>116.7K</td>
<td>61.4K</td>
<td>109.9K</td>
<td>77.2K</td>
<td>148.4K</td>
</tr>
<tr>
<td>CONV3</td>
<td>884.7K</td>
<td>309.7K</td>
<td>168.1K</td>
<td>241.2K</td>
<td>192.0K</td>
<td>299.3K</td>
</tr>
<tr>
<td>CONV4</td>
<td>663.5K</td>
<td>245.5K</td>
<td>132.7K</td>
<td>207.4K</td>
<td>182.6K</td>
<td>275.6K</td>
</tr>
<tr>
<td>CONV5</td>
<td>442.7K</td>
<td>163.7K</td>
<td>88.5K</td>
<td>134.7K</td>
<td>116.6K</td>
<td>197.1K</td>
</tr>
<tr>
<td>FC1</td>
<td>37.7M</td>
<td>3.40M</td>
<td>1.06M</td>
<td>0.763M</td>
<td>1.566M</td>
<td>0.781M</td>
</tr>
<tr>
<td>FC2</td>
<td>16.8M</td>
<td>1.51M</td>
<td>0.99M</td>
<td>1.070M</td>
<td>0.974M</td>
<td>0.650M</td>
</tr>
<tr>
<td>FC3</td>
<td>4.10M</td>
<td>1.02M</td>
<td>0.38M</td>
<td>0.505M</td>
<td>0.490M</td>
<td>0.472M</td>
</tr>
<tr>
<td>Total</td>
<td>60.9M</td>
<td>6.8M</td>
<td>2.9M</td>
<td>3.05M</td>
<td>3.62M</td>
<td>2.85M</td>
</tr>
</tbody>
</table>

Especially effective for large FC layers
Mix-Precision with Bit-Level Sparsity

- Selecting the optimal precision for each layer introduced a large and discrete design space.
- For a fixed-point quantized matrix, when can its precision be reduced?
  - MSB=0 for all elements: precision can reduce directly
    \[
    \begin{bmatrix}
    0 & 1 & 1 & 0 \\
    0 & 0 & 1 & 1 \\
    \end{bmatrix}
    \equiv
    \begin{bmatrix}
    1 & 1 & 0 \\
    0 & 1 & 1 \\
    \end{bmatrix}
    \]
  - LSB=0 for all elements: precision can be reduced with scaling factor 2
    \[
    \begin{bmatrix}
    1 & 0 & 1 & 0 \\
    0 & 1 & 0 & 0 \\
    \end{bmatrix}
    \equiv
    2 \times
    \begin{bmatrix}
    1 & 0 & 1 \\
    0 & 1 & 0 \\
    \end{bmatrix}
    \]
- MP quantization scheme can be explored by inducing structural bit-level sparsity

[ICLR’21, H. Yang et al.]
Mix-Precision with Bit-Level Sparsity

- Start with 8-bit quantized model
- Bit-level group LASSO:
- Overall training objective:
- Apply periodic re-quantization and precision adjustment throughout the training process

[ICLR’21, H. Yang et al.]
System-level Reliability Improvement

- ReSNA: ReRAM-based Stochastic-Noise-Aware Training

![Graphs and charts showing performance metrics such as Inferencing Accuracy, Area Overhead, Execution Time, and Energy Consumption.](image)
Robust and Efficient ReRAM-based System

ReSNA Training

DNN Parameters

Hardware Configurations

Design Objectives

Multi-Objective Optimization

Xiaoxuan Yang, et al., ICCAD 2021
Conclude & Our Lab
Conclusion

• The progress of hardware development needs to match up with the upscaling of DNN models at software level.

• A holistic scheme integrating the efforts on device, circuit and algorithm levels is important.

• Execution acceleration, energy efficiency, and design flexibility will greatly benefit from device-circuit-algorithm co-designs.
Thank you and Q&A