Science use-cases for real-time AI

- Collider on-detector readout and “trigger” at 40 MHz
- Accelerator control
- Neutrino physics
- Multi-messenger astronomy
- Electron & X-ray microscopy

And more…

w/MIT, UW, UIUC LIGO groups
w/Northwestern, JHU, Lehigh, ORNL

Discussed similar problems with other fields as well!

Neutrinos

Real-time filtering at the sub-millisecond scale required for next generation neutrino detectors to identify neutrinos from supernovae, $P \sim 25$ years

40 MHz $O(Pb/s)$

ASICs

CMS detector

L1 trigger: FPGA filter stack

DUNE v beam

Main Injector

Proton energy: 100 GeV

NuMI v beam

NOvA experiment

Booster v beam

MicroBooNE, ICARUS, BOREXINO

L1 trigger:

FPGA

filter stack

electron gun

magnetic lens

backscattered electron detector

specimen

secondary electron detector

stage

LIGO - A GIGANTIC INTERFEROMETER

1. Laser light is sent into the interferometer Michelson in pairs at the two arms.
2. A beam splitter splits the light and sends both identical beams along the two arms.
3. The light beams recombine and interfere.
4. A gravitational wave affects the interferometer's arms differentially, allowing us to detect the wave.

Overshot light is too broad for the detector.
Scientific ML challenges
AI model sizes

NLP model size is increasing exponentially

- GPT: 0.11B
- MegatronLM: 8.3B
- T-NLG: 17B
- GPT-2: 1.5B
- BERT: 0.34B
- Transformer: 0.05B
- GPT-3: 170B

Year
- 2017
- 2018
- 2020
- 2021
Codesign

- **Codesign**: intrinsic development loop between algorithm design, training, and implementation
- Compression
  - Maintain high performance while removing redundant operations
- Quantization
  - Reduce precision from 32-bit floating point to 16-bit, 8-bit, …
- Parallelization
  - Balance parallelization (how fast) with resources needed (how costly)
What is pruning?

Make neural network smaller by removing synapses and neurons

Optimal Brain Damage [LeCun et al., NeurIPS 1989]
Learning Both Weights and Connections for Efficient Neural Network [Han et al., NeurIPS 2015]
Pruning research

Learning both Weights and Connections for Efficient Neural Networks

Song Han
Stanford University
songhan@stanford.edu

Jeff Pool
NVIDIA
jpool@nvidia.com

John Tran
NVIDIA
johntran@nvidia.com

William J. Dally
Stanford University
NVIDIA
dally@stanford.edu

Optimal Brain Damage

Yann LeCun, John S. Denker and Sara A. Solla
AT&T Bell Laboratories, Holmdel, N. J. 07733
Pruning formalism

• In general, we could formulate the pruning as follows:

\[
\arg\min_{W_p} L(x; W_p)
\]

subject to

\[
\|W_p\|_0 < N
\]

• \(L\) represents the objective function for neural network training;

• \(x\) is input, \(W\) is original weights, \(W_p\) is pruned weights;

• \(\|W_p\|_0\) calculates the \#nonzeros in \(W_p\), and \(N\) is the target \#nonzeros.
Structured vs. unstructured pruning

- Unstructured pruning: removing some connections regardless of placement
  - Can potentially achieve higher performance for smaller size
- Structured pruning: removing all input/output connections of particular nodes
  - More regular structure easier to support in hardware architectures
Benchmark: Jet tagging MLP

Small NN benchmark correctly identifies particle “jets” 70-80% of the time.

16 inputs
64 nodes
32 nodes
32 nodes
5 outputs

hls4ml

<table>
<thead>
<tr>
<th>True label</th>
<th>Predicted label</th>
</tr>
</thead>
<tbody>
<tr>
<td>g</td>
<td>0.76 0.13 0.02 0.01 0.08</td>
</tr>
<tr>
<td>q</td>
<td>0.15 0.73 0.03 0.02 0.07</td>
</tr>
<tr>
<td>w</td>
<td>0.04 0.17 0.74 0.04 0.01</td>
</tr>
<tr>
<td>z</td>
<td>0.04 0.15 0.08 0.71 0.02</td>
</tr>
<tr>
<td>t</td>
<td>0.07 0.03 0.05 0.03 0.82</td>
</tr>
</tbody>
</table>

Normalized confusion matrix

$u,d$ or s jet
gluon jet
$W$ or $Z$ jet
top jet
Iterative magnitude-based pruning

- Train with $L_1$ regularization (down-weights unimportant synapses)

$$L_\lambda(w) = L(w) + \lambda \| w \|_1$$

$$\| w \|_1 = \sum_i |w_i|$$

- Remove smallest weights
- Iterate

70% REDUCTION OF WEIGHTS WITH NO LOSS IN PERF.

arXiv:1804.06913
Iterative magnitude-based pruning

- Train with $L_1$ regularization (down-weights unimportant synapses)

\[ L_\lambda(w) = L(w) + \lambda \|w\|_1 \]

\[ \|w\|_1 = \sum_i |w_i| \]

- Remove smallest weights
- Iterate

70% REDUCTION OF WEIGHTS WITH NO LOSS IN PERF.

arXiv:1804.06913
Pruning

1st iteration

2nd iteration

7th iteration

Train with L1

Prune

Retrain with L1

Prune

Retrain with L1

Prune

Retrain with L1

Prune
Pruning APIs

- TensorFlow API: [https://www.tensorflow.org/model_optimization/guide/pruning](https://www.tensorflow.org/model_optimization/guide/pruning)
- Sparsity schedule for gradual pruning
- Similar API for PyTorch: [https://pytorch.org/tutorials/intermediate/pruning_tutorial.html](https://pytorch.org/tutorials/intermediate/pruning_tutorial.html)

Figure 2: (a) The gradual sparsity function and exponentially decaying learning rate used for training sparse-InceptionV3 models. (b) Evolution of the model’s accuracy during the training process.

Table 1: Model size and accuracy tradeoff for sparse-InceptionV3

<table>
<thead>
<tr>
<th>Sparsity</th>
<th>NNZ params</th>
<th>Top-1 acc.</th>
<th>Top-5 acc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0%</td>
<td>27.1M</td>
<td>78.1%</td>
<td>94.3%</td>
</tr>
<tr>
<td>50%</td>
<td>13.6M</td>
<td>78.0%</td>
<td>94.2%</td>
</tr>
<tr>
<td>75%</td>
<td>6.8M</td>
<td>76.1%</td>
<td>93.2%</td>
</tr>
<tr>
<td>87.5%</td>
<td>3.3M</td>
<td>74.6%</td>
<td>92.5%</td>
</tr>
</tbody>
</table>

is to prune the network rapidly in the initial phase when the redundant connections are abundant and gradually reduce the number of weights being pruned each time as there are fewer and fewer weights remaining in the network, as illustrated in Figure 1. In the experimental results presented in this paper, pruning is initiated after the model has been trained for a few epochs or from a pre-trained model. This determines the value for the hyperparameter $t_0$. A suitable choice for $n$ is largely dependent on the learning rate schedule. Stochastic gradient descent (and its many variants) typically decay the learning rate during training, and we have observed that pruning in the presence of an exceedingly small learning rate makes it difficult for the subsequent training steps to recover from the loss in accuracy caused by forcing the weights to zero. At the same time, pruning with too high of a learning rate may mean pruning weights when the weights have not yet converged to a good solution, so it is important to choose the pruning schedule closely with the learning rate schedule.

Figure 2a shows the learning rate and the pruning schedule used for training sparse-InceptionV3 (Szegedy et al., 2016) models. All the convolutional layers in this model are pruned using the same sparsity function, and pruning occurs in the regime where the learning rate is still reasonably high to allow the network to heal from the pruning-induced damage. Figure 2b offers more insight into how this pruning scheme interacts with the training procedure. For the 87.5% sparse model, with the gradual increase in sparsity, there comes a point when the model suffers a near-catastrophic degradation, but recovers nearly just as quickly with continued training. This behavior is more pronounced in the models trained to have higher sparsity. Table 1 compares the performance of sparse-InceptionV3 models pruned to varying extents. As expected, there is a gradual degradation in the model quality as the sparsity increases. However, a 50% sparse model performs just as well as the baseline (0% sparsity), and there is only a 2% decrease in top-5 classification accuracy for the 87.5% sparse model which offers an 8x reduction in number of nonzero (NNZ) model parameters. Also note that since the weights are initialized randomly, the sparsity in the weight tensors does not exhibit any specific structure. Furthermore, the pruning method described here does not depend on any specific property of the network or the constituent layers, and can be extended directly to a wide-range of neural network architectures.
Aside: Lottery Ticket Hypothesis

• A randomly-initialized, dense neural network contains a subnetwork that is initialized such that—when trained in isolation—it can match the test accuracy of the original network after training for at most the same number of iterations.

![Diagram of Lottery Ticket Hypothesis]

- Randomly initialize weights and train
- Prune
- 90% accuracy

- Use same weight initialization and train
- 90% accuracy

- 90% accuracy

arXiv:1803.03635
Numeric data types: integer

- Unsigned Integer
  - n-bit Range: \([0, 2^n - 1]\)

- Signed Integer
  - Sign-Magnitude Representation
    - n-bit Range: \([-2^{n-1}, 2^{n-1} - 1]\)
    - Both 000...00 and 100...00 represent 0

- Two’s Complement Representation
  - n-bit Range: \([-2^{n-1}, 2^{n-1} - 1]\)
  - 000...00 represents 0
  - 100...00 represents \(-2^{n-1}\)
Numeric data types: fixed-point number

Fixed-Point Number

+ 0 0 1 1 0 0 0 1
-2^3 + 2^2 + 2^1 + 2^0 + 2^{-1} + 2^{-2} + 2^{-3} + 2^{-4} = 3.0625

× 2^{-4} = 49 × 0.0625 = 3.0625

(Using 2's complement representation)
Numeric data types: floating-point number

Example: 32-bit floating-point number in IEEE 754

\[ (-1)^{\text{sign}} \times (1 + \text{Fraction}) \times 2^{\text{Exponent} - 127} \]

(significant / mantissa)

0.265625 = 1.0625 \times 2^{-2} = (1 + 0.0625) \times 2^{125-127}
What is quantization?

Quantization is the process of constraining an input from a continuous or otherwise large set of values to a discrete set.

Images are in the public domain.

Wikipedia: Quantization
Quantization types

- Quantization: using reduced precision for parameters and operations
- Fixed-point precision
- Affine integer quantization
Affine integer quantization

An affine mapping of integers to real numbers $r = S(q - Z)$

Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference [Jacob et al., CVPR 2018]
Post-training quantization vs. quantization-aware training

<table>
<thead>
<tr>
<th></th>
<th>PTQ</th>
<th>QAT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Usually fast</td>
<td>Usually fast</td>
<td>Slow</td>
</tr>
<tr>
<td>No re-training of the model</td>
<td>Model needs to be trained/finetuned</td>
<td></td>
</tr>
<tr>
<td>Plug and play of quantization schemes</td>
<td>Plug and play of quantization schemes (requires re-training)</td>
<td></td>
</tr>
<tr>
<td>Less control over final accuracy of the model</td>
<td>More control over final accuracy since q-params are learned during training.</td>
<td></td>
</tr>
</tbody>
</table>
Post-training quantization

- General strategy: avoid overflows in integer bit then scan the decimal bit until reaching optimal performance

```
0101.1011101010
```

```
ap_fixed<width,integer>
```

---

Jennifer Ngadiuba - hls4ml: deep neural networks in FPGAs
25.04.2018
Quantization-aware training: how does it work?

- Fake quantization: using 32-bit floating-point math under the hood
- Straight-through estimator: during backpropagation, ignore quantization operation (treat as identity)
Quantization-aware training

- Full performance with 6 bits instead of 14 bits
- Much smaller fraction of resources
- Area & power scale quadratically with bit width

---

**Figure 2**: Area ($n$ vs. bitwidth ($m_b$) for a PE with a $w_b$-bit activations and $a_b$-bit weights requires $m_a$ bit operations. In particular, the multiplication of arithmetic “work” needed to calculate the entire network of $b$-bit number by $a$-bit activations and $b$-bit weights requires $m_n k$ bit operations, where $m_a$, $m_n$, and $m_k$ are, respectively, the number of input and output features of the layer. The formula takes into account the CNN topology and define the design rules of efficient implementation of quantized neural networks. The HCM metric assesses two and the communication complexity, which defines the memory access pattern and bandwidth. We describe the changes resulting from switching from a floating-point representation to a fixed-point one, and then present our computation and communication complexity metrics. All results for the fixed-point MAC operation access pattern and bandwidth. We describe the changes resulting from switching from a floating-point representation to a fixed-point one, and then present our computation and communication complexity metrics. All results for the fixed-point MAC operations, even for the same input bitwidth. To illustrate this fact, we generated two multipliers: one for 32-bit floating-point and the other one for 16-bit fixed-point numbers. The fixed-point MAC operations are very rare, however, it is often possible to reduce the accuracy of the network approximately eight time less area, gates, and power than the AI accelerators. Finally, Baskin et al. show that BOPS can be used as an estimator for the area required for a single PE with variable bitwidth, BOPS can be used to replicate the same individual PE design. For example, one PE with 1026 bit operations, where $b = \log_2(20)$. This shows that BOPS can be used as an estimator for the area required for a single PE with variable bitwidth, BOPS can be used to replicate the same individual PE design. For example, one PE with 1026 bit operations, where $b = \log_2(20)$. This shows that BOPS can be used as an estimator for the area required for a single PE with variable bitwidth, BOPS can be used to replicate the same individual PE design. For example, one PE with 1026 bit operations, where $b = \log_2(20)$. This shows that BOPS can be used as an estimator for the area required for a single PE with variable bitwidth, BOPS can be used to replicate the same individual PE design. For example, one PE with 1026 bit operations, where $b = \log_2(20)$. This shows that BOPS can be used as an estimator for the area required for a single PE with variable bitwidth, BOPS can be used to replicate the same individual PE design. For example, one PE with 1026 bit operations, where $b = \log_2(20)$. This shows that BOPS can be used as an estimator for the area required for a single PE with variable bitwidth, BOPS can be used to replicate the same individual PE design. For example, one PE with 1026 bit operations, where $b = \log_2(20)$. This shows that BOPS can be used as an estimator for the area required for a single PE with variable bitwidth, BOPS can be used to replicate the same individual PE design. For example, one PE with 1026 bit operations, where $b = \log_2(20)$. This shows that BOPS can be used as an estimator for the area required for a single PE with variable bitwidth, BOPS can be used to replicate the same individual PE design. For example, one PE with 1026 bit operations, where $b = \log_2(20)$. This shows that BOPS can be used as an estimator for the area required for a single PE with variable bitwidth, BOPS can be used to replicate the same individual PE design. For example, one PE with 1026 bit operations, where $b = \log_2(20)$. This shows that BOPS can be used as an estimator for the area required for a single PE with variable bitwidth, BOPS can be used to replicate the same individual PE design. For example, one PE with 1026 bit operations, where $b = \log_2(20)$. This shows that BOPS can be used as an estimator for the area required for a single PE with variable bitwidth, BOPS can be used to replicate the same individual PE design. For example, one PE with 1026 bit operations, where $b = \log_2(20)$. This shows that BOPS can be used as an estimator for the area required for a single PE with variable bitwidth, BOPS can be used to replicate the same individual PE design. For example, one PE with 1026 bit operations, where $b = \log_2(20)$. This shows that BOPS can be used as an estimator for the area required for a single PE with variable bitwidth, BOPS can be used to replicate the same individual PE design. For example, one PE with 1026 bit operations, where $b = \log_2(20)$. This shows that BOPS can be used as an estimator for the area required for a single PE with variable bitwidth, BOPS can be used to replicate the same individual PE design. For example, one PE with 1026 bit operations, where $b = \log_2(20)$. This shows that BOPS can be used as an estimator for the area required for a single PE with variable bitwidth, BOPS can be used to replicate the same individual PE design. For example, one PE with 1026 bit operations, where $b = \log_2(20)$. This shows that BOPS can be used as an estimator for the area required for a single PE with variable bitwidth, BOPS can be used to replicate the same individual PE design. For example, one PE with 1026 bit operations, where $b = \log_2(20)$. This shows that BOPS can be used as an estimator for the area required for a single PE with variable bitwidth, BOPS can be used to replicate the same individual PE design. For example, one PE with 1026 bit operations, where $b = \log_2(20)$. This shows that BOPS can be used as an estimator for the area required for a single PE with variable bitwidth, BOPS can be used to replicate the same individual PE design. For example, one PE with 1026 bit operations, where $b = \log_2(20)$. This shows that BOPS can be used as an estimator for the area required for a single PE with variable bitwidth, BOPS can be used to replicate the same individual PE design. For example, one PE with 1026 bit operations, where $b = \log_2(20)$. This shows that BOPS can be used as an estimator for the area required for a single PE with variable bitwidth, BOPS can be used to replicate the same individual PE design. For example, one PE with 1026 bit operations, where $b = \log_2(20)$. This shows that BOPS can be used as an estimator for the area required for a single PE with variable bitwidth, BOPS can be used to replicate the same individual PE design. For example, one PE with 1026 bit operations, where $b = \log_2(20)$. This shows that BOPS can be used as an estimator for the area required for a single PE with variable bitwidth, BOPS can be used to replicate the same individual PE design. For example, one PE with 1026 bit operations, where $b = \log_2(20)$. This shows that BOPS can be used as an estimator for the area required for a single PE with variable bitwidth, BOPS can be used to replicate the same individual PE design. For example, one PE with 1026 bit operations, where $b = \log_2(20)$. This shows that BOPS can be used as an estimator for the area required for a single PE with variable bitwidth, BOPS can be used to replicate the same individual PE design. For example, one PE with 1026 bit operations, where $b = \log_2(20)$. This shows that BOPS can be used as an estimator for the area required for a single PE with variable bitwidth, BOPS can be used to replicate the same individual PE design. For example, one PE with 1026 bit operations, where $b = \log_2(20)$. This shows that BOPS can be used as an estimator for the area required for a single PE with variable bitwidth, BOPS can be used to replicate the same individual PE design. For example, one PE with 1026 bit operations, where $b = \log_2(20)$. This shows that BOPS can be used as an estimator for the area required for a single PE with variable bitwidth, BOPS can be used to replicate the same individual PE design. For example, one PE with 1026 bit operations, where $b = \log_2(20)$. This shows that BOPS can be used as an estimator for the area required for a single PE with variable bitwidth, BOPS can be used to replicate the same individual PE design. For example, one PE with 1026 bit operations, where $b = \log_2(20)$. This shows that BOPS can be used as an estimator for the area required for a single PE with variable bitwidth, BOPS can be used to replicate the same individual PE design. For example, one PE with 1026 bit operations, where $b = \log_2(20)$. This shows that BOPS can be used as an estimator for the area required for a single PE with variable bitwidth, BOPS can be used to replicate the same individual PE design. For example, one PE with 1026 bit operations, where $b = \log_2(20)$. This shows that BOPS can be used as an estimator for the area required for a single PE with variable bitwidth, BOPS can be used to replicate the same individual PE design. For example, one PE with 1026 bit operations, where $b = \log_2(20)$. This shows that BOPS can be used as an estimator for the area required for a single PE with variable bitwidth, BOPS can be used to replicate the same individual PE design. For example, one PE with 1026 bit operations, where $b = \log_2(20)$. This shows that BOPS can be used as an estimator for the area required for a single PE with variable bitwidth, BOPS can be used to replicate the same individual PE design. For example, one PE with 1026 bit operations, where $b = \log_2(20)$. This shows that BOPS can be used as an estimator for the area required for a single PE with variable bitwidth, BOPS can be used to replicate the same individual PE design. For example, one PE with 1026 bit operations, where $b = \log_2(20)$. This shows that BOPS can be used as an estimator for the area required for a single PE with variable bitwidth, BOPS can be used to replicate the same individual PE design. For example, one PE with 1026 bit operations, where $b = \log_2(20)$. This shows that BOPS can be used as an estimator for the area required for a single PE with variable bitwidth, BOPS can be used to replicate the same individual PE design. For example, one PE with 1026 bit operations, where $b = \log_2(20)$. This shows that BOPS can be used as an estimator for the area required for a single PE with variable bitwidth, BOPS can be used to replicate the same individual PE design. For example, one PE with 1026 bit operations, where $b = \log_2(20)$. This shows that BOPS can be used as an estimator for the area required for a single PE with variable bitwidth, BOPS can be used to replicate the same individual PE design. For example, one PE with 1026 bit operations, where $b = \log_2(20)$.
Pruning + quantization-aware training  

- Quantization-aware pruning (QAP): iterative pruning can further reduce the hardware computational complexity of a quantized model
- After QAP, the 6-bit, 80% pruned model achieves a factor of 50 reduction in BOPs compared to the 32-bit, unpruned model
- Study using Brevitas

Bit operations (BOPs) definition:
arXiv:1804.10969
Hessian-aware quantization (HAWQ)

- Hessian of loss can provide additional guidance about quantization!
- Flat loss landscape: Lower bit width
- Sharp loss landscape: Higher bit width
COMPUTING HARDWARE ALTERNATIVES

- FPGAs
- CPUs
- GPUs
- ASICs

Flexibility vs. Efficiency
LHC event processing

Compute Latency

1 ns 1 μs 1 ms 1 s

L1 Trigger

750 kHz

High-Level Trigger

7.5 kHz 1 MB/evt

Offline

ASICs

FPGAs

CPUs

GPUs

FPGAs

CPUs

GPUs

Challenges:

Each collision produces $O(10^3)$ particles
The detectors have $O(10^8)$ sensors
Extreme data rates of $O(100 \text{ TB/s})$
APPLICATION: MEASURE MUONS AT 40 MHZ

- NN measures muon momentum
- 3× reduction in the trigger rate for NN!
- Fits within L1 trigger latency (240 ns!) and FPGA resource requirements (less then 30%)

Figure 3.33: Left: endcap trigger rate comparison of the Phase-1 EMTF and the Phase-2 EMTF++ algorithms as a function of $p_T$ threshold for events with 200 average pileup. Right: Trigger rate comparison as a function of PU for a $p_T > 20$ GeV threshold.

The same 6 $D_f$ zones are retained for a total of 54 patterns per 0.5-degree in $f$, as for the prompt muon patterns. Figure 3.31 (right) shows these patterns.

The TP information in the stations from stubs that satisfy a displaced pattern are input to a NN that in this case has been trained to perform a regression that returns simultaneously values for $1/p_T$ and $d_0$ of displaced muons. The NN configuration used is the same as that for prompt muons, using 3 hidden layers with 30/25/20 nodes each. Batch normalization is inserted after each layer, including the input layer. A total of 23 inputs are used in the NN, these are:

- 6 $D_q$ quantities between stations: S1-S2, S1-S3, S1-S4, S2-S3, S2-S4, S3-S4
- 4 bend angles: set to zero if no CSC stub is found and only RPC stub is used
- For ME1 only: 1 bit for front or back chambers and 1 bit for inner or outer $h$ ring
- 1 track $q$ taken from stub coordinate in ME2, ME3, ME4 (in this priority)
- 4 RPC bits indicating if ME or RE stub was used in each station (S1, S2, S3, S4)

At the time of this writing, information from the new Phase-2 detectors (GE1/1, GE2/1, ME0, iRPC) has not been incorporated into the study, and neither has the more precise CSC bend information described above. As such, this study is geared towards possible implementation of this algorithm during Run-3. An update to incorporate new Phase-2 detector information is in progress. The already positive conclusions on triggering on standalone displaced muons in the endcap with only the Phase-1 detectors, as shown below, is expected to improve significantly when all Phase-2 information is included.

Figure 3.34 shows, for events with single muons and no pileup, the $q/p_T$ and the $d_0$ resolutions as determined by the NN estimate of these quantities. The $p_T$ resolution is about 60%, which is large compared to the 20% resolution obtained from EMTF++ for prompt muons. A bias towards underestimating the $p_T$ can be observed. However, the $d_0$ resolution is very good, $\ll 5$ cm. Figure 3.35 shows the trigger rates of the displaced muon algorithm for PU 200 events. In order to keep the rates at approximately the same 10 kHz level as those from prompt muons, reasonable L1 thresholds of, for example, $p_T > 20$ GeV and $|d_0| > 20$ cm can be applied.
Next time

- Knowledge distillation