Optimizing Power and Performance for Machine Learning at the Edge: Model Deployment Overview

Lingchuan Meng, Principal Engineer, Arm
Naveen Suda, Principal Engineer, Arm

October 20, 2020
# AI Virtual Tech Talks Series

<table>
<thead>
<tr>
<th>Date</th>
<th>Title</th>
<th>Host</th>
</tr>
</thead>
<tbody>
<tr>
<td>October 20, 2020</td>
<td>Optimizing Power and Performance For Machine Learning at the Edge - Model Deployment Overview</td>
<td>Arm</td>
</tr>
<tr>
<td>November 3, 2020</td>
<td>Small is big: Making Deep Neural Nets faster, smaller and energy-efficient on low power hardware</td>
<td>DeepLite</td>
</tr>
<tr>
<td>November 17, 2020</td>
<td>The Smart City in Motion - AI in intelligent transportation</td>
<td>Clever Devices, NXP Semiconductor, Arcturus</td>
</tr>
</tbody>
</table>

Visit: developer.arm.com/solutions/machine-learning-on-arm/ai-virtual-tech-talks
Presenters

Lingchuan Meng, Principal Engineer, Arm

Naveen Suda, Principal Engineer, Arm
ML on the Edge - Challenges

• Edge device constraints for deploying ML algorithms
  • Limited memory
    – Flash (32 kB - few MB)
    – SRAM (16 kB - 512 kB)
  • Limited compute capability (100 MHz - 1 GHz)

• Hardware/software features
  • Compression HW: pruning, clustering, etc.
  • Mixed precision: 8-bit, 16-bit, etc.
  • Algorithmic: Winograd, etc.
  • Layer fusion: conv-add-pool-relu, etc.

ML solutions = Model Optimization → Software → Hardware
End-to-end Technology Exploration

ML Networks
- Vision
- Voice
- Vibration

Model Architecture Optimizations
- Domain expert
- Rule-based replacement
- Neural architecture search

Model Deployment Optimizations
- Pruning
- Quantization
- Clustering
- Algorithms

Software
- Compiler
- Drivers
- Libraries
- Optimized kernels
- Scheduling
- Memory allocation
- Operator coverage

Hardware
- Models
- RTL
- FPGA
- Sparsity
- Low-Precision Arithmetic
- Compression
- HW Algorithms

PPAB
- Perf
- Power
- Area
- Bandwidth

Algo/SW/HW co-dev
Model Deployment Optimizations
Deployment Optimization Flow

**Collaborative Optimizations**

- **Trained models** → **Algorithmic Optimizations** → **Parameter Optimization** → **Structured Pruning** → **Magnitude Pruning** → **Weight Clustering** → **candidate models** → **NetOpt models** → **Equalization** → **Fold batch-norms** → **Fuse layers** → **Quantization** → **Fine-tuning** → **DepOpt models**

- **Network Optimizations**

- **Deployment Optimizations**

- **HW**

© 2020 Arm Limited
Overview of Technologies

<table>
<thead>
<tr>
<th>Optimization Collaboration</th>
<th>Cascaded Optimization with Attribute Preservation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hyper-param Auto-tuning</td>
<td></td>
</tr>
<tr>
<td>Heuristics</td>
<td></td>
</tr>
<tr>
<td>Reinforcement Learning</td>
<td></td>
</tr>
<tr>
<td>Simulated Annealing</td>
<td></td>
</tr>
<tr>
<td>Optimization Collaboration</td>
<td></td>
</tr>
<tr>
<td>Magnitude Pruning</td>
<td></td>
</tr>
<tr>
<td>Clustering</td>
<td></td>
</tr>
<tr>
<td>Post-training Quant</td>
<td></td>
</tr>
<tr>
<td>Mixed-precision Quant</td>
<td></td>
</tr>
<tr>
<td>Optimizations</td>
<td></td>
</tr>
<tr>
<td>Structured pruning</td>
<td></td>
</tr>
<tr>
<td>Channel pruning</td>
<td></td>
</tr>
<tr>
<td>Winograd</td>
<td></td>
</tr>
<tr>
<td>Sub-graph Substitution</td>
<td></td>
</tr>
<tr>
<td>Utilities</td>
<td></td>
</tr>
<tr>
<td>Graph Editing &amp; Node Selection</td>
<td></td>
</tr>
<tr>
<td>Dynamic Range Equalization</td>
<td></td>
</tr>
<tr>
<td>Sensitivity Analysis</td>
<td></td>
</tr>
<tr>
<td>Perf Model Integration</td>
<td></td>
</tr>
</tbody>
</table>

CPU MCU  GPU  NPU uNPU
## Optimization Results

<table>
<thead>
<tr>
<th>Model</th>
<th>Sparsity</th>
<th>Accuracy Δ</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inception V3</td>
<td>50%</td>
<td>+0.1%</td>
</tr>
<tr>
<td>ResNet 50</td>
<td>50%</td>
<td>+0.7%</td>
</tr>
<tr>
<td>VGG 16</td>
<td>50%</td>
<td>+1.6%</td>
</tr>
<tr>
<td>MobileNet V1</td>
<td>50%</td>
<td>-0.9%</td>
</tr>
<tr>
<td>Wav2Letter</td>
<td>50%</td>
<td>-1.34%</td>
</tr>
<tr>
<td>DS-CNN Large</td>
<td>80%</td>
<td>-0.5%</td>
</tr>
</tbody>
</table>

*unstructured pruning*

### Key Metrics

<table>
<thead>
<tr>
<th></th>
<th>Before</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error Rate</td>
<td>+0.93%</td>
<td>+0.31%</td>
</tr>
<tr>
<td>Infer Time</td>
<td>-32%</td>
<td>-41%</td>
</tr>
<tr>
<td>SRAM BW</td>
<td>-43%</td>
<td>-40%</td>
</tr>
<tr>
<td>Flash BW</td>
<td>-60%</td>
<td>-44%</td>
</tr>
</tbody>
</table>

Overview of Pruning Techniques

Magnitude Pruning

Channel Pruning

Structured Pruning


Pruning

• Inducing sparsity to overly-parametrized models
  • Improve model compression and computation efficiency

• Structure
  • Unstructured: irregular locations of zeros
  • Structured: pre-defined patterns of zeros

• Spatial granularity
  • Layer / filter / kernel / weight
  • Compressibility vs. acceleration

• Techniques
  • Magnitude / Variational dropout / Regularization

• Challenges
  • Accuracy degradation
  • High sparsity → better performance?
Pruning – Key Concepts

• Pruning schedule
  • Pruning induces damages to model
  • Increase sparsity gradually
  • Strategies: inverse power/linear/cosine

• Distribution of sparsity
  • Not all layers are equal
  • Uniform: same sparsity for all layers
  • Heuristic: sparsity \( \propto \) # parameters
  • Reinforcement-learning (RL)

• Hardware-aware hyper-parameter tuning
  • Tuning for a single optimization
  • Joint tuning for multiple optimizations
Hyper-Parameter Tuning

Deterministic

• Uniform
  • Same sparsity for all prunable layers

• Heuristic
  • Per-layer target sparsity:
  \[
  \alpha = \frac{pr \cdot \sum|var_i|}{\sum(|var_i| \cdot \log|var_i|)}
  \]
  • Dynamically increase pruning ratio during training
  \[
  \bar{pr} = pr \cdot \left(1 - \left(1 - \frac{t - t_0}{n\Delta t}\right)^3\right)
  \]

Reinforcement Learning


Hardware-Aware Hyper-Parameter Tuning

- Optimizing for accuracy + hardware metrics
  - HW metrics: latency, compression, bandwidth ...
  - Multi-objective reward/fitness functions
  - Joint tuning for multiple optimizations
    - Larger search space and better results

![Graphs showing multi-objective reward, accuracy, and inference/sec](image)
Convolution with Sparse Tensors

- Accelerating sparse matrix multiply
  - Sparse weights + dense activations
  - Dense math primitives → sparse primitives
  - Vector loads of activations
  - Randomly-accessed values cached in L1
  - Prefetching activations to reduce cache misses
  - Block-structured sparsity for additional speedup

<table>
<thead>
<tr>
<th>Model</th>
<th>Width</th>
<th>Top-1</th>
<th>Mega FLOPs</th>
<th>Mega Params</th>
<th>Time SD835</th>
<th>Time SD670</th>
<th>Time Wasm</th>
</tr>
</thead>
<tbody>
<tr>
<td>MBv1</td>
<td>Dense</td>
<td>1.0</td>
<td>70.9</td>
<td>1120</td>
<td>4.30</td>
<td>125</td>
<td>106</td>
</tr>
<tr>
<td></td>
<td>Sparse</td>
<td>1.4</td>
<td>72.0</td>
<td>268</td>
<td>2.28</td>
<td>58</td>
<td>64</td>
</tr>
<tr>
<td>MBv1</td>
<td>Dense</td>
<td>0.75</td>
<td>68.4</td>
<td>636</td>
<td>2.59</td>
<td>73</td>
<td>64</td>
</tr>
<tr>
<td></td>
<td>Sparse</td>
<td>1.0</td>
<td>68.4</td>
<td>146</td>
<td>1.48</td>
<td>31</td>
<td>34</td>
</tr>
</tbody>
</table>

Algorithmic Optimizations

Complex-domain Winograd

8-bit Winograd

- int-F(4x4) complex
- float-F(6x6)
- Vector
  - int-F(4x4)
  - int-F(2x2)
- float-im2col
- Scalar
  - int-F(4x4)
  - int-F(2x2)

2.24X

Accuracy

<table>
<thead>
<tr>
<th></th>
<th>Training</th>
<th>Top-1</th>
<th>Top-5</th>
</tr>
</thead>
<tbody>
<tr>
<td>FP32</td>
<td></td>
<td>76.94%</td>
<td>93.40%</td>
</tr>
<tr>
<td>Int8 (FQ)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Im2Col</td>
<td>X</td>
<td>76.38%</td>
<td>93.13%</td>
</tr>
<tr>
<td>F(4x4) Real</td>
<td>X</td>
<td>0.52%</td>
<td>1.79%</td>
</tr>
<tr>
<td>F(4x4) Complex</td>
<td>√</td>
<td>60.49%</td>
<td>82.86%</td>
</tr>
<tr>
<td>F(4x4) Complex</td>
<td>X</td>
<td>74.86%</td>
<td>92.43%</td>
</tr>
<tr>
<td>F(4x4) Complex</td>
<td>√</td>
<td>76.27%</td>
<td>93.10%</td>
</tr>
</tbody>
</table>

Inception V3


NCNN: https://github.com/Tencent/ncnn

Speedup vs NCNN 2x2

- VGG 16: 94.55%
- ResNet 18: 21.13%
- GoogleNet: 12.82%
- SqueezeNet: 8.86%

© 2020 Arm Limited
Quantization

- Storing and computing with tensors at lower bitwidths
  - Typically FP32 -> INT8: $x_{FP32} = scale \cdot (x_{INT8} - zero\_point)$
  - 4x savings in model size and memory bandwidth
  - Inference speedup: 2-4x
  - More aggressive quantization in active research

- Granularity: per-layer vs. per-channel

- Symmetric vs. asymmetric
  - Weights: symmetric with zero_point=0
  - Activations: asymmetric

- Finding optimal quantization ranges
  - Balancing range vs. resolution
  - Techniques: minimize Quantization error, KL-divergence, etc.
Quantization Workflow

1. **Trained model (floating point)**
2. **Weight quantization**
   - Find optimal quantization ranges for weights and evaluate model with quantized weights
3. **Collect statistics of activations.**
   - Insert histogram Ops and collect the distribution of activations.
4. **Find act. quantization ranges**
   - Find optimal quantization thresholds for activations
5. **Quantize and evaluate**
   - Evaluate weight/activation quantized model
6. **Quantized model**
   - Optionally finetune if training data is available
   - => Quantization aware training (QAT)
7. **Optional – finetuning**
   - Finetuned quantized model
Quantizing Activation Nodes

- Simulate quantization in forward pass.
- Straight-through-estimator (STE) in backward pass during QAT.

- Quantize all the operators in a model.
- Fuse layers wherever possible, before quantization.
Mixed-Precision Quantization

- Some layers are less sensitive to aggressive quantization.
- How to find the optimal bit-width per layer?
- A solution: Sensitivity-based mixed precision quantization
  - Find lowest bit-width without significant accuracy drop.
  - Consider the cascaded effect of quantization error from layer-to-layer.
  - Start from the largest layer, so it is compressed the most.

**Mobilenet V2**
- Average bitwidth: 4.5 bits
- 2% accuracy drop (without retraining)
- Fine-tuning recovered 1.5% accuracy
Clustering: Non-Uniform Quantization

- Non-uniform quantization yields smaller quantization error than uniform quantization.
- Better weight compression – especially for large layers.
- K-means clustering algorithm to find initial cluster centroids.
- Fine-tune the cluster centroids with clustering constraints in the graph.
- Preserve sparsity during fine-tuning using sparsity masks.
Collaborative Optimizations

Weights distribution

-3.0 0 3.1

Pruning

-3.2 -0.2 0.2 3.5

50% sparsity

Clustering (8 clusters)

-2.4 0 2.7

8-unique \textbf{fp32} values

Quantize

-110 0 127

50% sparsity

8-unique \textbf{int8} values

Cluster preserving

Centroids (c)

W \xrightarrow{\text{Argmin}(W-c)} \text{indices}

Gather \rightarrow \text{Clustered weights}

Sparsity preserving

\text{Mul}

\text{Fake Quant}

Quantization

\text{Conv/FC}

y

x

\begin{align*}
\text{sparsity mask} & : \text{sparsity mask} \\
(W_{\text{min}}, W_{\text{max}}), \text{num_bits} & : (W_{\text{min}}, W_{\text{max}}), \text{num_bits}
\end{align*}
Model Architecture Optimizations
Efficient Network Building Blocks

• Arithmetic reduction by operator decomposition/approximation
  • Depthwise convolution – Mobilenet-V1
  • Inverted bottleneck with residual – Mobilenet-V2

• Operator sparsification
  • sparsity → performance?
  • Replace dense ops with sparse ops

• Asymptotically-faster operators
  • Winograd convolution
  • FFT
Efficiency is Target-Specific

- Hardware utilization matters!
  - Inverted bottleneck conv block
  - Fused inverted bottleneck conv block
    - 1x1 conv + 3x3 depthwise -> 3x3 conv
    - More trainable parameters
    - Better HW utilization (hence a good latency-accuracy trade-off)
  - 3x3 vs. 5x5 convolution
    - 5x5 convolution leads to 2.78x increase of MACs and parameters
    - Only a 35% runtime increase
    - Good trade-off for more trainable parameters at a marginal latency cost

Efficiency is More than Conv/FC

- Layer normalization and Gaussian Error LU (GELU) impact latency
- Layer normalization replaced by element-wise linear transform
  - \( \text{NoNorm}(h) = \gamma \circ h + \beta \)
- GELU replaced by ReLU

---

<table>
<thead>
<tr>
<th>Model</th>
<th>#Params</th>
<th>#FLOPS</th>
<th>Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>MobileBERT_{Tiny}</td>
<td>15.1M</td>
<td>3.1B</td>
<td>40 ms</td>
</tr>
<tr>
<td>MobileBERT</td>
<td>25.3M</td>
<td>5.7B</td>
<td>62 ms</td>
</tr>
<tr>
<td>MobileBERT w/o OPT</td>
<td>25.3M</td>
<td>5.7B</td>
<td>192 ms</td>
</tr>
</tbody>
</table>

---

Hardware-aware Neural Architecture Search (NAS)

• Diversity in ML Hardware – CPUs, GPUs and NPUs
  • Different programming models, compute capabilities, memory organization
  • Same neural network architecture cannot map efficiently across multiple HW

• Awareness of the target HW architecture by NN architectures
  • Hand tuning model architectures
  • Automated neural architecture search (NAS)

• Search objectives
  • Latency proxies: # MACs, # parameters (NASNet)
  • Hardware performance model (MNASNet, ProxylessNAS, FBNet)

• State-of-the-art models searched by NAS
  • MobileNet v3, FBNet, EfficientNet, MobileBERT
NAS – Key Concepts

- **Search space**
  - Chain-structured
  - Architecture template
  - Cell-based

- **Search strategy**
  - Reinforcement learning
  - Gradient-based methods
  - Evolutionary algorithms
  - Random search with rejection sampling

- **Performance estimation**
  - Accuracy estimation
  - Latency estimation
NAS Search Space

Chain-structured

Architecture template

Cell-based

Summary

• Resource constraints and diverse SW + HW features require co-development
• Model optimizations bridge the gap between models + SW + HW
• Efficient building blocks and architectures for higher accuracy and performance
• NAS efficiently navigates in design spaces to automate model design
• New opportunities in joint optimization of NAS, pruning, and quantization
Thank you!

Tweet us: @ArmSoftwareDev

Check out our Arm YouTube channel and our Arm Software Developers YouTube channel

Signup now for our next AI Virtual Tech Talk here

Don’t forget to fill out our survey to be in with a chance of winning an Arduino Nano 33 BLE board
Thank You
Danke
Merci
谢谢
ありがとう
Gracias
Kiitos
감사합니다
धन्यवाद
شكرًا
תודה