The Edge of Machine Learning

Bottom-up or Top-down?

Aditya Kusupati
University of Washington
Edge Machine Learning – Objectives

• To build a library of machine learning algorithms
  • Which can be trained in the cloud
  • But which will run on tiny IoT devices

ARM Cortex M0+
Microsoft’s EdgeML Library

- Compact tree, kNN and RNN algorithms for classification, regression, ranking, time series etc.,

Bonsai
ICML’17

ProtoNN
ICML’17

EMI-RNN
NeurIPS’18

FastGRNN
NeurIPS’18

https://github.com/Microsoft/EdgeML
Recognizing “Hey, Cortana” in 1 KB

- Uncompressed FastGRNN outperforms state-of-the-art RNNs
- FastGRNN matches state-of-the-art RNN accuracies

**F1 Score**

**Model Size (KB)**

Proposed:
- FastGRNN

Existing:
- SpectralRNN

- RNN
- UGRNN
- GRU
- LSTM
- SpectralRNN
Soft Threshold Weight Reparameterization for Learnable Sparsity

Aditya Kusupati
Vivek Ramanujan*, Raghav Somani*, Mitchell Wortsman*
Prateek Jain, Sham Kakade and Ali Farhadi
Motivation

- **Deep Neural Networks**
  - Highly accurate
  - Millions of parameters & Billions of FLOPs
  - Expensive to deploy

- **Sparsity**
  - Reduces model size & inference cost
  - Maintains accuracy
  - Deployment on CPUs & weak single-core devices
Motivation

• Existing sparsification methods
  • Focus on model size vs accuracy – *very little on inference FLOPs*
  • Global, uniform or heuristic sparsity budget across layers

<table>
<thead>
<tr>
<th>Layer 1</th>
<th>Layer 2</th>
<th>Layer 3</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td># Params</td>
<td>20</td>
<td>100</td>
<td>1000</td>
</tr>
<tr>
<td>FLOPs</td>
<td>100K</td>
<td>100K</td>
<td>50K</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sparsity – Method 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td># Params</td>
<td>20</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>FLOPs</td>
<td>100K</td>
<td>100K</td>
<td>5K</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sparsity – Method 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td># Params</td>
<td>10</td>
<td>10</td>
<td>200</td>
</tr>
<tr>
<td>FLOPs</td>
<td>50K</td>
<td>10K</td>
<td>10K</td>
</tr>
</tbody>
</table>
Motivation

• Non-uniform sparsity budget – Layer-wise
  • Very hard to search in deep networks
  • Sweet spot – Accuracy vs FLOPs vs Sparsity
• Existing techniques
  • Heuristics – increase FLOPs
  • Use RL – expensive to train

“Can we design a robust efficient method to learn non-uniform sparsity budget across layers?”
Overview

- **STR** – **Soft Threshold Reparameterization**

\[ STR(W_l, \alpha_l) = \text{sign}(W_l) \cdot \text{ReLU}(|W_l| - \alpha_l) \]

- Learns layer-wise non-uniform sparsity budgets
  - Same model size; Better accuracy; Lower inference FLOPs
  - SOTA on ResNet50 & MobileNetV1 for ImageNet-1K
  - Boosts accuracy by up to 10% in ultra-sparse (98-99%) regime

- Extensions to structured, global & per-weight (mask-learning) sparsity
Existing Methods

- SOTA; Dense training cost
  - Dense-to-sparse training
    - Uniform sparsity
      - Gradual Magnitude Pruning (GMP)
    - Non-uniform sparsity
      - Heuristics – ERK
      - Global Pruning/Sparsity
      - STR - some gains from sparse-to-sparse
      - Learnable sparsity?
  - Hybrid
    - DNW & DPF
  - Sparse-to-sparse training
    - Non-uniform sparsity
      - DSR, SNFS, RigL etc.,
      - Heuristics – ERK
      - Re-allocation using magnitude/gradient
  - Hard to train; Lower training cost
STR - Method

\[ HT(x, \alpha) = \begin{cases} 
  x; & |x| > \alpha \\
  0; & |x| \leq \alpha 
\end{cases} \]

\[ ST(x, \alpha) = \begin{cases} 
  x - \alpha; & x > \alpha \\
  0; & |x| \leq \alpha \\
  x + \alpha; & x < -\alpha 
\end{cases} \]
STR - Method

\[ ST(x, \alpha) = \text{sign}(x) \cdot \text{ReLU}(|x| - \alpha) \]
\[ = \text{sign}(x) \cdot \text{ReLU}(|x| - g(s)) \]

\[ L\text{-layer DNN, } \mathcal{W} = [W_l]_{l=1}^{L}, s = [s_l]_{l=1}^{L} \text{ and a function } g(.) \]

\[ S_g(W_l, s_l) = \text{sign}(W_l) \cdot \text{ReLU}(|W_l| - g(s_l)) \]

\[ \mathcal{W} \leftarrow S_g(\mathcal{W}, s) \]
STR - Training

\[
\min_{\mathcal{W}, s} \mathcal{L}(\mathcal{S}_g(\mathcal{W}, s), \mathcal{D}) + \lambda \sum_{l=1}^{L} (|\mathbf{W}_l|^2 + |s_l|^2)
\]

• Regular training with reparameterized weights \( \mathcal{S}_g(\mathcal{W}, s) \)

• Same weight-decay parameter (\( \lambda \)) for both (\( \mathcal{W}, s \))
  • Controls the overall sparsity

• Initialize \( s; g(s) \approx 0 \)
  • Finer sparsity and dense training control

• Choice of \( g(. ) \)
  • *Unstructured sparsity*: Sigmoid
  • *Structured sparsity*: Exponential
\[ \mathbf{W}_l^{(t+1)} \leftarrow (1 - \eta_t \cdot \lambda) \mathbf{W}_l^{(t)} - \eta_t \nabla S_g(\mathbf{W}_l, s_l) \mathcal{L}(S_g(\mathbf{W}^{(t)}, s), \mathcal{D}) \odot \nabla \mathbf{W}_l S_g(\mathbf{W}_l, s_l), \]

\[ \mathbf{W}_l^{(t+1)} \leftarrow (1 - \eta_t \cdot \lambda) \mathbf{W}_l^{(t)} - \eta_t \nabla S_g(\mathbf{W}_l, s_l) \mathcal{L}(S_g(\mathbf{W}^{(t)}, s), \mathcal{D}) \odot 1 \left\{ S_g(\mathbf{W}_l^{(t)}, s_l) \neq 0 \right\}, \]

\[ \nabla_{s_l} \mathcal{L}(\widetilde{\mathbf{W}}(s_l)) = \nabla_{s_l} \mathcal{L}(S_g(\mathbf{W}_l, s_l)) = -g'(s_l) \mathcal{P}(\mathbf{W}_l, g(s_l)) \]

\[ \mathcal{P}(\mathbf{W}_l, g(s_l)) := \left\langle \nabla_{\widetilde{\mathbf{W}}(s_l)} \mathcal{L}(\widetilde{\mathbf{W}}(s_l)), \text{sign}(\mathbf{W}_l) \odot 1 \left\{ \widetilde{\mathbf{W}}_l(s_l) \neq 0 \right\} \right\rangle \]

\[ s_l^{(t+1)} \leftarrow s_l^{(t)} + \eta_t g'(s_l^{(t)}) \mathcal{P}(\mathbf{W}_l^{(t)}, g(s_l^{(t)})) - \eta_t \lambda s_l^{(t)} \]
• STR learns the SOTA hand-crafted heuristic for threshold

![Threshold vs Epochs for Layer 10 - 90% sparse ResNet50 on ImageNet-1K](image1)

• STR learns unique threshold values per-layer

![Layer-wise threshold – 90% sparse ResNet50 on ImageNet-1K](image2)
STR - Training

- STR learns the SOTA hand-crafted heuristic of GMP

![Graph showing overall sparsity vs Epochs for 90% sparse ResNet50 on ImageNet-1K.](image)

- STR learns diverse non-uniform layer-wise sparsities

![Graph showing layer-wise sparsity for 90% sparse ResNet50 on ImageNet-1K.](image)
STR - Experiments

• Unstructured sparsity - CNNs
  • *Dataset*: ImageNet-1K
  • *Models*: ResNet50 & MobileNetV1
  • *Sparsity range*: 80 - 99%
    • Ultra-sparse regime: 98 - 99%

• Structured sparsity – Low rank in RNNs
  • *Datasets*: Google-12 *(keyword spotting)*, HAR-2 *(activity recognition)*
  • *Model*: FastGRNN

• Additional
  • Transfer of learnt budgets to other sparsification techniques
  • STR for global, per-weight sparsity & filter/kernel pruning
Unstructured vs Structured Sparsity

• Unstructured sparsity
  • Typically magnitude based pruning with global or layer-wise thresholds

• Structured sparsity
  • Low-rank & neuron/filter/kernel pruning
STR Unstructured Sparsity: ResNet50

- STR requires 20% lesser FLOPs with same accuracy for 80-95% sparsity
- STR achieves 10% higher accuracy than baselines in 98-99% regime
STR Unstructured Sparsity: MobileNetV1

- STR maintains accuracy for 75% sparsity with 62M lesser FLOPs
- STR has \(~50\%\) lesser FLOPs for 90% sparsity with same accuracy
STR Sparsity Budget: ResNet50

- STR learns sparser initial layers than the non-uniform sparsity baselines
- STR makes last layers denser than all baselines
- STR produces sparser backbones for transfer learning
- STR adjusts the FLOPs across layers such that it has lower total inference cost than the baselines
STR Sparsity Budget: MobileNetV1

- STR automatically keeps depth-wise separable conv layers denser than rest of the layers.
- STR’s budget results in 50% lesser FLOPs than GMP.
STR Budget Transfer: ResNet50

- Gradual Magnitude Pruning (GMP) – Zhu & Gupta 2017

<table>
<thead>
<tr>
<th>Method</th>
<th>Top-1 Acc (%)</th>
<th>Params</th>
<th>Sparsity (%)</th>
<th>FLOPs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uniform</td>
<td>73.91</td>
<td>2.56M</td>
<td>90.00</td>
<td>409M</td>
</tr>
<tr>
<td>Budget from STR</td>
<td><strong>74.13</strong></td>
<td>2.49M</td>
<td><strong>90.23</strong></td>
<td><strong>343M</strong></td>
</tr>
<tr>
<td>Uniform</td>
<td>57.90</td>
<td>0.51M</td>
<td>98.00</td>
<td>82M</td>
</tr>
<tr>
<td>Budget from STR</td>
<td><strong>59.47</strong></td>
<td>0.50M</td>
<td><strong>98.05</strong></td>
<td><strong>73M</strong></td>
</tr>
</tbody>
</table>

- Discovering Neural Wirings (DNW) – Wortsman et al., NeurIPS 2019

<table>
<thead>
<tr>
<th>Method</th>
<th>Top-1 Acc (%)</th>
<th>Params</th>
<th>Sparsity (%)</th>
<th>FLOPs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uniform</td>
<td>74.00</td>
<td>2.56M</td>
<td>90.00</td>
<td>409M</td>
</tr>
<tr>
<td>ERK</td>
<td><strong>74.10</strong></td>
<td>2.56M</td>
<td>90.00</td>
<td>960M</td>
</tr>
<tr>
<td>Budget from STR</td>
<td>74.01</td>
<td>2.49M</td>
<td><strong>90.23</strong></td>
<td><strong>343M</strong></td>
</tr>
<tr>
<td>Uniform</td>
<td>68.30</td>
<td>1.28M</td>
<td>95.00</td>
<td>204M</td>
</tr>
<tr>
<td>Budget from STR</td>
<td><strong>69.72</strong></td>
<td>1.33M</td>
<td>94.80</td>
<td>182M</td>
</tr>
<tr>
<td>Budget from STR</td>
<td>68.01</td>
<td>1.24M</td>
<td><strong>95.15</strong></td>
<td><strong>162M</strong></td>
</tr>
</tbody>
</table>
Algorithm 1 PyTorch code for STRConv with per-layer threshold.

```python
import torch
import torch.nn as nn
import torch.nn.functional as F

from args import args as parser_args

def softThreshold(x, s, g=torch.sigmoid):
    # STR on a weight x (can be a tensor) with "s" (typically a scalar, but can be a tensor) with function "g".
    return torch.sign(x) * torch.relu(torch.abs(x) - g(s))

class STRConv(nn.Conv2d):  # Overloaded Conv2d which can replace nn.Conv2d
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        # "g" can be chosen appropriately, but torch.sigmoid works fine.
        self.g = torch.sigmoid
        # parser_args gets arguments from command line. sInitValue is the initialization of "s" for all layers. It
        # can take in different values per-layer as well.
        self.s = nn.Parameter(parser_args.sInitValue*torch.ones([1, 1]))
        # "s" can be per-layer (a scalar), global (a shared scalar across layers), per-channel/filter (a vector)
        # or per individual weight (a tensor of the size self.weight). All the experiments use per-layer "s" (a
        # scalar) in the paper.

    def forward(self, x):
        self.sparseWeight = softThreshold(self.weight, self.s, self.g)
        # Parameters except "x" and "self.sparseWeight" can be chosen appropriately. All the experiments use
        # default PyTorch arguments.
        x = F.conv2d(x, self.sparseWeight, self.bias, self.stride, self.padding, self.dilation, self.groups)
        return x

# FC layer is implemented as a 1x1 Conv2d and STRConv is used for FC layer as well.
```
STR Structured Sparsity: Low rank

Typical low-rank parameterization

Train with STR on $\Sigma$
More STR Adaptations

- Neuron/Filter/Kernel pruning

\[
\mathcal{S}_g(W_l, s) = \text{sign}(W_l) \cdot \text{ReLU}(|W_l| - g(s))
\]

- Global sparsity/pruning

\[
\mathcal{S}_g(W_l, S_l) = \text{sign}(W_l) \cdot \text{ReLU}(|W_l| - g(S_l))
\]
STR – Critical Design Choices

• Weight-decay $\lambda$
  • Controls overall sparsity
  • Larger $\lambda \rightarrow$ higher sparsity at the cost of some instability

• Initialization of $s_l$
  • Controls finer sparsity exploration
  • Controls duration of dense training

• Careful choice of $g(\cdot)$
  • Drives the training dynamics
  • Better functions which consistently revive dead weights
STR - Conclusions

• STR enables stable end-to-end training (with no additional cost) to obtain sparse & accurate DNNs

• STR efficiently learns per-layer sparsity budgets
  • Reduces FLOPs by up to 50% for 80-95% sparsity
  • Up to 10% more accurate than baselines for 98-99% sparsity
  • Transferable to other sparsification techniques

• Future work
  • Formulation to explicitly minimize FLOPs
  • Stronger guarantees in standard sparse regression setting

• Code, pretrained models and sparsity budgets available at
  https://github.com/RAIVNLab/STR