IEEE ICASSP 2020 Tutorial on Distributed and Efficient Deep Learning

Outline of tutorial

1. Introduction (WS)
2. Model Compression & Efficient Deep Learning (WS)
3. (Virtual) Coffee Break
4. Distributed & Federated Learning (FS)

Wojciech Samek (Fraunhofer HHI)
Felix Sattler (Fraunhofer HHI)
Part I: Introduction

Wojciech Samek & Felix Sattler
Deep Learning "Revolution"

Ingredients for the success:
1. Large volumes of data
2. Large (Deep) Models
3. Large Computing Power
Complexity of DNN is Growing
Complexity of DNN is Growing

We need to design efficient architectures and techniques to reduce the model size.
Large Computational Resources Needed

Common carbon footprint benchmarks

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>CO2 Equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Roundtrip flight b/w NY and SF (1 passenger)</td>
<td>1,984</td>
</tr>
<tr>
<td>Human life (avg. 1 year)</td>
<td>11,023</td>
</tr>
<tr>
<td>American life (avg. 1 year)</td>
<td>36,156</td>
</tr>
<tr>
<td>US car including fuel (avg. 1 lifetime)</td>
<td>126,000</td>
</tr>
<tr>
<td>Transformer (213M parameters) w/ neural architecture search</td>
<td>626,155</td>
</tr>
</tbody>
</table>

Chart: MIT Technology Review • Source: Strubell et al. • Created with Datawrapper
We need techniques to reduce the computational complexity at *training time*.
Processing at the “Edge”

Cameras and radar generate ~6 gigabytes of data every 30 seconds.

Self-driving car prototypes use approximately 2,500 Watts of computing power.

Generates wasted heat and some prototypes need water-cooling!

[slide from V. Sze]
Processing at the “Edge”

On-device deep learning

Distributed Data & Privacy

Latency & bandwidth constraints
We need techniques to reduce the computational complexity at *inference time* (i.e., storage, memory, energy, runtime)
We need techniques to robustly train our models on *distributed data* in a *privacy-preserving* manner.
Large Interest in Academia & Industry
Large Interest in Academia & Industry

![Bar Chart showing publications mentioning FL from 2015 to 2020]

- Year 2015: 0 publications
- Year 2016: 10 publications
- Year 2017: 20 publications
- Year 2018: 30 publications
- Year 2019: 1200 publications
- Year 2020: ? publications

**PUBLICATIONS**

Federated Learning for Mobile Keyboard Prediction

**BANKING & PAYMENTS**

Tencent’s WeBank applying “federated learning” in AI

China’s first mobile bank, Tencent’s WeBank, is partnering with a H.K. startup to access sources of data.

Artificial Intelligence / Machine Learning

How Apple personalizes Siri without hoovering up your data

The tech giant is using privacy-preserving machine learning to improve its voice assistant while keeping your data on your phone.

by Karen Hao

Dec 13, 2019

---

IEEE ICASSP 2020 Tutorial on Distributed and Efficient Deep Learning
Large Interest in Academia & Industry

Apple packed an AI chip into the iPhone X

Intel, Qualcomm, Google, and NVIDIA Race to Develop AI Chips and Platforms

source: https://github.com/basicmi/AI-Chip
Large Interest in Academia & Industry

Standard on "Compression of Neural Networks for Multimedia Content Description and Analysis"
Part II: Model Compression & Efficient Deep Learning

Wojciech Samek & Felix Sattler
This part will discuss how to reduce the complexity of DNNs by **model compression** and **efficient representation**.

| 1. Background: Pruning, Quantization & Encoding |
| 2. Finding Efficient Architectures |
| 3. Compression Techniques |
| 4. Efficient Neural Network Representation |
Background: Pruning, Quantization & Encoding
Source Coding

Source coding is a subfield of information theory that studies the properties of so-called codes.

The primary task of a source codec is to represent a signal with the minimum number of (binary) symbols without exceeding an “acceptable level of distortion”, which is determined by the application.
Source Coding

**Goal:** Minimize the rate-distortion objective:

\[
C^* = \arg \min_C \mathbb{E}_{P(w)} \left[ D(w, q) + \lambda L_C(b) \right]
\]

where \( b = (B \circ Q)(w) \) and \( q = (Q^{-1} \circ Q)(w) \).
Lossless Coding

\[ q = (Q^{-1} \circ Q)(w) = w \ \forall w \]

The rate-distortion objective simplifies into finding a binarizer \( B^* \) that maximally compresses the input samples.

Information theory already makes concrete statements regarding the minimum information contained in a probability source.

The minimum information required to fully represent a sample \( w \) that has probability \( P(w) \) is of \(-P \log_2 P(w)\) bits (Shannon).
Lossless Coding

Consequently, the entropy \( H_P(\mathbb{W}) = \sum_{w \in \mathbb{W}} -P(w) \log_2 P(w) \) states the minimum average number of bits required to represent any element \( w \in \mathbb{W} \subset \mathbb{R}^n \).

**Fundamental theorem of lossless coding**

\[
H_P(\mathbb{W}) \leq \bar{L}_C(\mathbb{W}), \quad \forall C
\]

where \( \bar{L}_C(\mathbb{W}) = \sum_{w \in \mathbb{W}} P(w)L_C(w) \) is the average code-length that any code \( C \) assigns to each element \( w \in \mathbb{W} \).
Lossless Coding

We know of the existence of codes that are able to reach the average code-length, up to only 1 bit of redundancy.

\[ \exists C : H_P(\mathbb{W}) \leq \bar{L}_C(\mathbb{W}) < H_P(\mathbb{W}) + 1 \]

Moreover, we even know how to build them.

**Example:** Huffman coding
Lossless Coding: Huffman Codes

\[
\begin{align*}
P(7) &= 0.29 & '1' & P = 0.57 & '11' \\
P(6) &= 0.28 & '0' & '1' & '10' \\
P(5) &= 0.16 & '1' & '0' & '01' \\
P(4) &= 0.14 & '1' & '0' & '001' \\
P(3) &= 0.07 & '1' & '0' & '0001' \\
P(2) &= 0.03 & '1' & '0' & '00001' \\
P(1) &= 0.02 & '1' & '0' & '000001' \\
P(0) &= 0.01 & '0' & '0' & '000000' \\
\end{align*}
\]
Lossless Coding: Huffman Codes

However, Huffman codes can be very inefficient in practice since the Huffman-tree grows very quickly for large input dimensions $n$.

**Scalar Huffman codes**

\[
H_P(W) \leq \bar{L}_{SH}(W) < H_P(W) + n
\]

**Arithmetic codes**

Produces only up to two bits more than the minimum possible code length of an $n$-long random process.
Lossless Coding: Arithmetic Coding

Arithmetic coding consist of expressing a particular sequence of samples of an n-long random process as a so-called coding interval.

\[
\begin{array}{ccccccc}
\text{W}_0 & 0 & 1 & 1 & 1 & 1 & 1 \\
\text{W}_1 & 0 & 0 & 1 & 1 & 1 & 1 \\
\text{W}_2 & 0 & 0 & 1 & 1 & 1 & 1 \\
\text{W}_3 & 0 & 0 & 1 & 1 & 1 & 1 \\
\text{W}_4 & 0 & 0 & 1 & 1 & 1 & 1 \\
\end{array}
\]

100 (index '4')

encode

'10111'
Lossless Coding: Desired Properties

**Universality:** The code should have a mechanism that allows it to adapt its probability model to a wide range of different types of input distributions, in a sample-efficient manner.

**Minimal redundancy:** The code should produce binary representations of minimal redundancy with regards to its probability estimate.

**High efficiency:** The code should have high coding efficiency, meaning that encoding/decoding should have high throughput.
Lossless Coding: Universal Codes

We implicitly assumed that the decoder knows the joint probability distribution of the input source.

This is not the case in many real world scenarios. Hence, in such cases one usually relies on so called universal codes.

They basically apply the following principle:
1) start with a general, prior probability model.
2) update the model upon seeing data.
3) encode the input samples with regards to the updated probability model.

\[
\tilde{L}_C(W) \geq H_{P,P_{\text{Dec}}}(W) = H_P(W) + D_{KL}(P \parallel P_{\text{Dec}})
\]
Lossy Coding

Finding the optimal code is in most cases NP-hard

\[ C^* = \arg \min_C \mathbb{E}_{P(w)} \left[ D(w, q) + \lambda L_C(b) \right] \]

Idea: Fix the binarization map B by selecting a particular (universal) lossless code. Then just need to find a scalar quantizer

\[ (Q, Q^{-1})^* = \arg \min_{(Q, Q^{-1})} \mathbb{E}_{P(w_j)} \left[ D(w_j, q_j) + \lambda L_Q(b_j) \right] \]

(we ignore vector quantizers here, which measure the distortion in the respective vector space by grouping a sequence of input samples together)
Lossy Coding: Scalar Lloyd algorithm

It approximates the average code-length of the quantized samples with the entropy of their empirical probability mass distribution (EPMD).

\[ L_C(b_j) = - \log_2 P_{\text{EPMD}}(q_j) \]

k-means clustering:
1) assign to closest cluster centers (= quantize)
2) update cluster centers (= update reconstruction values)
Lossy Coding: Uniform quantization

\[ C^* = \arg \min_C \mathbb{E}_{P(w)} [D(w, q) + \lambda L_C(b)] \]

Only minimize the distortion and ignore the rate term.

\[ \lambda = 0 \]

Only one step size parameter:

\[ q_k = \Delta I_k \]
Lossy Coding: CABAC-based RD-quantization

\[ C^* = \arg \min_C \mathbb{E}_{P(w)} [D(w, q) + \lambda L_C(b)] \]

Select Context Adaptive Binary Arithmetic Coder (CABAC) as our universal code.

Then we can trivially minimize by sequentially quantizing the input samples.
Source Coding vs. NN Coding

Signal compression
Distortion between elements (e.g. pixel values)

\[
\arg \min_{(Q, Q^{-1})} D(w_j, q_j) + \lambda L(b)
\]
Source Coding vs. NN Coding

Signal compression
Distortion between elements
(e.g. pixel values)

Neural network compression
Distortion between function of elements
(e.g. prediction outputs)
NN Coding

In NN coding, things are more complicated:
- complex distortion term (non-linear accumulation of errors)
- no clear structure in NN weights (e.g. in video high correlation between frames and neighboring pixels)
- more flexibility (e.g. fine-tuning, sparsification, structural changes)

In the extreme case, the whole training process can be regarded as NN coding (MDL principle)

$$W^* = \min_W \left( -\log_2 p(Y|X, W) + \alpha \right)$$

[encoding prediction error] + [encoding model]

[Wiedemann et al. 2018]
NN Coding

is developing a standard on "Compression of Neural Networks for Multimedia Content Description and Analysis"
Finding Efficient Architectures
Pruning Techniques

before pruning

pruning synapses

pruning neurons

after pruning
Pruning Techniques

Network Pruning:

1. Given a pre-trained model in the target domain
2. Define a pruning criterion
3. Repeatedly prune the network as follows:
   i. For each layer,
      a. For each element (weight/filter), evaluate the importance according to the pruning criterion (compute magnitudes)
      b. optional: Globally scale the magnitudes with regularization (e.g. $l_p$-norm)
   ii. Sort the magnitudes for all the layers throughout the network
   iii. Prune the least important elements and their inputs and outputs
   iv. optional: Further fine-tune to compensate performance degradation
4. Stop pruning if the model is reduced to a desired amount of model size or performance
Pruning Techniques: Different Criteria

**Taylor expansion**: Leverage a second-order Taylor expansion based on the Hessian matrix of the loss function to select parameters for deletion (optimal brain damage and optimal brain surgeon)

**Gradient**: A sparsified back propagation approach for neural network training using the magnitude of the gradient to find essential and non-essential features.

**Weight**: Prune the weights whose magnitude is below a certain threshold and to subsequently fine-tune with a $l_1$-norm regularization.

**XAI-based**: LRP decomposes a classification decision into contributions called "relevances" of each network element to the overall classification score.
Layer-wise Relevance Propagation is a general approach to explain predictions of AI.

\[ \sum_i R_i = f(x) \]

Layer-wise Relevance Propagation is a general approach to explain predictions of AI.

[Bach et al., PLOS ONE, 2015]
Results with Fine-Tuning

Pruning with fine-tuning. [Yeom et al., arXiv, 2019]
Results without Fine-Tuning

[Yeom et al., arXiv, 2019]
Distilling & Neural Architecture Search

[source: https://towardsdatascience.com/knowledge-distillation-simplified-dd4973dbc764]

[source: Elsken et al. 2019]
Compression Techniques
From Source Coding to NN Coding

\[(Q, Q^{-1})^* = \arg \min_{(Q, Q^{-1})} \sum_{(x,y) \in D} L(y'', y') + \lambda L_Q(b)\]

\[y' \sim P(y'|x, w) \quad y'' \sim P(y''|x, q)\]

0.1
0.05
0.85
0.8
0
0.05
From Source Coding to NN Coding

\[(Q, Q^{-1})^* = \arg \min_{(Q, Q^{-1})} \sum_{(x, y) \in D} L(y'', y') + \lambda L_Q(b)\]

\[(Q, Q^{-1})^* = \arg \min_{(Q, Q^{-1})} \sum_{(x, y) \in D} D_{KL}(y'' || y') + \lambda L_Q(b)\]

Use KL-divergence as distortion measure
From Source Coding to NN Coding

$$(Q, Q^{-1})^* = \arg \min_{(Q, Q^{-1})} \sum_{(x, y) \in \mathcal{D}} \mathcal{L}(y'', y') + \lambda L_Q(b)$$

$$(Q, Q^{-1})^* = \arg \min_{(Q, Q^{-1})} \sum_{(x, y) \in \mathcal{D}} D_{KL}(y'' || y') + \lambda L_Q(b)$$

$$(Q, Q^{-1})^* = \min_{(Q, Q^{-1})} (q - w) F (q - w)^T + \lambda L_Q(b)$$

If the output distributions do not differ too much, we can approximate KL with the Fisher Information Matrix (FIM)
From Source Coding to NN Coding

\[(Q, Q^{-1})^* = \arg\min_{(Q, Q^{-1})} \sum_{(x, y) \in \mathcal{D}} \mathcal{L}(y'', y') + \lambda L_Q(b)\]

\[(Q, Q^{-1})^* = \arg\min_{(Q, Q^{-1})} \sum_{(x, y) \in \mathcal{D}} D_{KL}(y'' || y') + \lambda L_Q(b)\]

\[(Q, Q^{-1})^* = \min_{(Q, Q^{-1})} (q - w)F(q - w)^T + \lambda L_Q(b)\]

\[(Q, Q^{-1})^* = \arg\min_{(Q, Q^{-1})} F_i(q_i - w_i)^2 + \lambda L_Q(b)\]

Approximate FIM by only its diagonal elements
DeepCABAC: Weighted RD-based Quantization + CABAC

https://github.com/fraunhoferhhi/DeepCABAC

DeepCABAC-v1

Parametrize each weight parameter as Gaussian. $F_i = 1/\sigma_i$

$q_k = \Delta I_k$

[Wiedemann et al. 2019, ODML-CDNNR]

best paper award
DeepCABAC: Uniform Quantization + CABAC

https://github.com/fraunhoferhhi/DeepCABAC

DeepCABAC-v3

\[ F_j = 1 \ \forall j \quad \lambda = 0 \]

best paper award
Properties of CABAC

- **Binarization**: represents each unique input value as a sequence of binary decisions.
- **Context modelling**: probability model for each decision, which is updated on-the-fly by the local statistics of the data -> universality.
- **Arithmetic coding**: arithmetic coding for each bit -> minimal redundancy + high efficiency

Examples:
- $1 \rightarrow 100$
- $-4 \rightarrow 111101$
- $7 \rightarrow 10111010$
The first $n+2$ bits allow to adapt to any type of shape around 0 since they are encoded with a context model. The remainder can only approximate the shape by a step-like distribution, since they are encoded with an Exponential-Golomb where the fixed-length parts are encoded without a context model.
## Some Results

<table>
<thead>
<tr>
<th>Sparse Models (sparsity [%])</th>
<th>Org. Acc. Top1 [%]</th>
<th>Os_size [MB]</th>
<th>DeepCABAC (acc. [%])</th>
</tr>
</thead>
<tbody>
<tr>
<td>VGG16 (9.85)</td>
<td>69.43</td>
<td>553.43</td>
<td>1.57 (69.43)</td>
</tr>
<tr>
<td>ResNet50 (74.12)</td>
<td>74.09</td>
<td>102.23</td>
<td>4.74 (73.65)</td>
</tr>
<tr>
<td>Small-VGG16 (7.57)</td>
<td>91.35</td>
<td>60.01</td>
<td>1.6 (91.00)</td>
</tr>
<tr>
<td>LeNet5 (1.90)</td>
<td>99.22</td>
<td>1.72</td>
<td>0.72 (99.16)</td>
</tr>
</tbody>
</table>
### Some Results

<table>
<thead>
<tr>
<th>Sparse Models (sparsity [%])</th>
<th>Org. Acc. Top1 [%]</th>
<th>Os_size [MB]</th>
<th>DeepCABAC (acc. [%])</th>
</tr>
</thead>
<tbody>
<tr>
<td>VGG16</td>
<td>69.43</td>
<td>553.43</td>
<td>1.57</td>
</tr>
</tbody>
</table>

VGG16 **553.4MB** -> **8.7MB** at an acc. **69.43%**

ResNet50 **102.2MB**-> **4.85MB** at an acc. **73.65%**

<table>
<thead>
<tr>
<th>Small-VGG16 (7.57)</th>
<th>91.35</th>
<th>60.01</th>
<th>1.6 (91.00)</th>
</tr>
</thead>
<tbody>
<tr>
<td>LeNet5 (1.90)</td>
<td>99.22</td>
<td>1.72</td>
<td><strong>0.72</strong> (99.16)</td>
</tr>
</tbody>
</table>
Some Results
Some Results
Some Results
Efficient Neural Network Representation
Efficient Representation

Goal: Find a representation for the neural network, which is:
1) efficient with regard to storage / memory
2) efficient with regard to inference complexity
3) efficient with regard to energy consumption
Fixed-Point Neural Nets

Reduce Precision

32-bit float: 1010010100000000010100000000100
8-bit fixed: 01100110
Binary: 0

Advantages

- Arithmetic with lower bit-depth is faster
- From 32-bits to 8-bits, we get (almost) 4x reduction in memory
- Lower bit-widths we can squeeze more data into caches/registers
- Floating point arithmetic not always be supported on microcontrollers on some ultra low-power embedded devices

No guarantee that DNN algorithm designer will use a given approach. Need flexible hardware!
Efficient Representation Format

Goal: Find a representation for the weight matrices of a neural network, which is:
1) efficient with regard to storage
2) efficient with regard to inference complexity
Efficient Representation Format

\[
M = \begin{pmatrix}
0 & 3 & 0 & 2 & 4 & 0 & 0 & 2 & 3 & 4 & 0 & 4 \\
4 & 4 & 0 & 0 & 0 & 4 & 0 & 0 & 4 & 4 & 0 & 4 \\
4 & 0 & 3 & 4 & 0 & 0 & 0 & 4 & 0 & 2 & 0 & 0 \\
0 & 0 & 4 & 4 & 4 & 0 & 3 & 4 & 4 & 0 & 0 & 0 \\
0 & 4 & 4 & 4 & 0 & 0 & 4 & 0 & 4 & 0 & 0 & 0
\end{pmatrix}
\]

Storage requirements: 60 entries  dense format
Matrix Formats: Dense Format

\[
M = \begin{pmatrix}
0 & 3 & 0 & 2 & 4 & 0 & 0 & 2 & 3 & 4 & 0 & 4 \\
4 & 4 & 0 & 0 & 0 & 4 & 0 & 0 & 4 & 4 & 0 & 4 \\
4 & 0 & 3 & 4 & 0 & 0 & 0 & 4 & 0 & 2 & 0 & 0 \\
0 & 0 & 0 & 4 & 4 & 4 & 0 & 3 & 4 & 4 & 0 & 0 \\
0 & 4 & 4 & 0 & 0 & 4 & 0 & 4 & 0 & 0 & 0 & 0
\end{pmatrix}
\]

Storage requirements: 60 entries

Scalar product (second row \( M \), vector \( a \)):
- 24 load
- 12 multiply
- 11 add
- 1 write operations
Matrix Formats: Sparse Format

\[ W : [3, 2, 4, 2, 3, 4, 4, 4, 4, 4, 4, 4, 3, 
     4, 4, 2, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4] \]

\[ \text{colI} : [1, 3, 4, 7, 8, 9, 11, 0, 1, 5, 8, 9, 11, 0, 
     2, 3, 7, 9, 3, 4, 5, 7, 8, 9, 1, 2, 5, 7] \]

\[ \text{rowPtr} : [0, 7, 13, 18, 24, 28] \]

**sparse format**

Storage requirements: 62 entries

Scalar product (second row M, vector a):
- 20 load
- 6 multiply
- 5 add
- 1 write operations
Matrix Formats: Compressed Entropy Row Format

Compressed neural networks have weight sharing property.

Trick:

$$z^l_i = \sum_{j}^{M} w^l_{i,j} a^l_{j-1} \quad \rightarrow \quad z^l_i = \sum_{k}^{w} w^l_{k} \sum_{j \in J^l_{ik}} a^l_{j-1}$$
Matrix Formats: Compressed Entropy Row Format

\[ \Omega : [0, 4, 3, 2] \]
\[ \text{colI} : [4, 9, 11, 1, 8, 3, 7, 0, 1, 5, 8, 9, 11, 0, 3, 7, 2, 9, 3, 4, 5, 8, 9, 7, 1, 2, 5, 7] \]
\[ \Omega P : [0, 3, 5, 7, 13, 16, 17, 18, 23, 24, 28] \]
\[ \text{rowPtr} : [0, 3, 4, 7, 9, 10] \]

CER format

Storage requirements: 49 entries

Scalar product (second row \( M \), vector \( a \)):
- 17 load
- 1 multiply
- 5 add
- 1 write operations

\[ 4(a_1 + a_2 + a_6 + a_9 + a_{10} + a_{12}) \]

[Wiedemann et al. 2020, IEEE TNNLS]
Storage Efficiency

\[
\begin{pmatrix}
0 & 3 & 0 & 2 & 4 & 0 & 0 & 2 & 3 & 4 & 0 & 4 \\
4 & 4 & 0 & 0 & 4 & 0 & 0 & 4 & 4 & 0 & 4 \\
4 & 0 & 3 & 4 & 0 & 0 & 0 & 4 & 0 & 2 & 0 \\
0 & 0 & 0 & 4 & 4 & 4 & 0 & 3 & 3 & 4 & 0 \\
0 & 4 & 4 & 0 & 0 & 4 & 0 & 4 & 0 & 0 & 0
\end{pmatrix}
\]

\[
\text{Dense}
\]

\[
\text{CER/CSER}
\]

\[
\text{Sparse}
\]

\[
H = -\log_2 p_0
\]

[Wiedemann et al. 2020, IEEE TNNLS]
Results

Compressed AlexNet after converting its weight matrices into the different data structures.

[Wiedemann et al. 2020, IEEE TNNLS]
Conclusion

Efficiency in training and efficiency in inference.
Many different techniques to compress NN.
Different options (e.g. fine-tuning, structural changes, NAS).
Hardware co-design is crucial.
MPEG standardization.
References

Neural Network Compression


http://dx.doi.org/10.1109/IJCNN.2019.8852119
References

Efficient Deep Learning
http://dx.doi.org/10.1109/TNNLS.2019.2910073


Federated Learning


Federated Learning

http://dx.doi.org/10.1109/ICASSP40776.2020.9054676

http://dx.doi.org/10.1109/IJCNN.2019.8852172
Slides and Papers available at www.federated-ml.org