Vers un apprentissage profondément plus économique

Tricks, Theory, Measures and Hardware

Loustau Sébastien

joint work with Yanis Chaigneau, Matthieu François, Paul Gay, Simon Lebeaud, Jordy Palafox, Fatou Kiné Sow et Nicolas Tirel

31-08-2022
The computer bounce effect

**Deep neural networks are energy hungry and growing fast**

Al is being powered by the explosive growth of deep neural networks

Figure 1: Exponential growth of Deep Learning models
[Fournarakis, 2021a]
Figure 2: Carbon footprint comparative study between a neural network and activities [Han, 2021].

The impact of Machine Learning is non negligible → Reduce the size of models!
1. Tricks and limits

2. More maths

3. AIPowerMeter

4. Tiny ML

5. Conclusion and Perspectives
1 Tricks and limits
   Quantization
   Pruning
   Early Exits

2 More maths

3 AIPowerMeter

4 Tiny ML

5 Conclusion and Perspectives
1 Tricks and limits
   Quantization
   Pruning
   Early Exits

2 More maths

3 AIPowerMeter

4 Tiny ML

5 Conclusion and Perspectives
Quantization

Low bitwidth approaches

- Originally from [Courbariaux et al., 2015] - BinaryConnect,
- Weights and I/O in [Bulat and Tzimiropoulos, 2019] - XNOR-nets ++,
- Quantization in [Zhou et al., 2016, Fournarakis, 2021a].
BNN’s details

Figure 3: CNN and Binarization [Yuan and Agaian, 2021]
Backpropagation of a BNN

Figure 4: STE estimator in the backpropagation

Figure source:
Limits of BNN’s

- STE estimator and continuous gradient accumulations,
- No gain observed in [Courbariaux et al., 2015] or [Bulat and Tzimiropoulos, 2019],
Limits of BNN’s

- STE estimator and continuous gradient accumulations,
- No gain observed in [Courbariaux et al., 2015] or [Bulat and Tzimiropoulos, 2019],
- Larq Compute Engine to the rescue.

Figure 5: Larq Compute Engine workflow from training to deployment
Principle

• train your model on a standard CPU/GPU,
• convert the model in tflite format,
• a python module (GitHub link here) based on a C++ routine is used (tiling, vectorization and parallelism) for fast inference.

More details on a blog post here.
Larq Compute Engine gain

\[ h \times w \times \text{in} \times \text{out} \text{ convolutions are:} \]

- (A) \(56 \times 56 \times 64 \times 64\)
- (B) \(28 \times 28 \times 128 \times 128\)
- (C) \(14 \times 14 \times 256 \times 256\)
- (D) \(7 \times 7 \times 256 \times 256\)
1 Tricks and limits
   Quantization
   Pruning
   Early Exits

2 More maths

3 AIPowerMeter

4 Tiny ML

5 Conclusion and Perspectives
Pruning - origin

Originated in Leo Breiman’s book Classification And Regression Tree (1984):

- build a maximal decision tree $T_{\text{max}}$
- solve the following optimization problem:

$$\arg \min_{T \subset T_{\text{max}}} \left\{ \sum_{\text{node} \in T} \sum_{x_i \in \text{node}} (y_i - \bar{y}_{\text{node}})^2 + \alpha \|T\|_0 \right\}$$

- goal: reduce overfitting!
Pruning NNs

**no sparsity**  
output  
input

**connection sparsity**  
output  
input

**node sparsity**  
output  
input

**layer sparsity**  
output  
input

**combined sparsity**  
output  
input
Pruning NNs - pruned literature

• originated in [Mozer and Smolensky, 1988] with relevance coefficients and Optimal Brain Damage in [LeCun et al., 1990] : **pruning after training**

• recent advances in [Lee et al., 2018], [de Jorge et al., 2020] : **pruning at init**

• statistical approach for layer sparsity or **pruning during training** in [Hebiri and Lederer, 2020, Bellec et al., 2018].
We test a Python implementation of SNIP.

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Dataset</th>
<th>Pruning?</th>
<th>Parameters</th>
<th>Time (hh:mm:ss)</th>
<th>Max precision (%)</th>
<th>Total consumption (Wh)</th>
</tr>
</thead>
<tbody>
<tr>
<td>vgg-D</td>
<td>CIFAR-10</td>
<td>no</td>
<td>15,239,872</td>
<td>1:40:18</td>
<td>93.55</td>
<td>785</td>
</tr>
<tr>
<td></td>
<td></td>
<td>yes (95%)</td>
<td>761,994</td>
<td>1:39:03</td>
<td>93.13</td>
<td>771</td>
</tr>
<tr>
<td>LeNet-5-Caffe</td>
<td>MNIST</td>
<td>no</td>
<td>430,500</td>
<td>30:18</td>
<td>99.42</td>
<td>145.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>yes (98%)</td>
<td>8,610</td>
<td>28:26</td>
<td>99.15</td>
<td>145.28</td>
</tr>
</tbody>
</table>

No gain since units are zeroing.
We test PyTorch Sparse library for sparse matrix multiplication.
Pytorch Sparse with 1% of sparsity
SNIP with PyTorch.Sparse

Speedup with pruned models, using sparseLinear

Accuracy (5 epochs)

Size linear layers
cuSparse

The cuSparse 11.4.0 release provides “a new high-performance block sparse matrix multiplication routine” for newer GPUs, with the help of the Blocked-ELL format.

read the cuSPARSE Library doc from Nvidia.
Comparison between cuSparse SpMM and cuBlas hGeMM

**cuSparse speedup in comparison with cuBlas, for different matrix sizes, as a function of the sparsity**

- n = 1024
- n = 2048
- n = 4096
- n = 8192
- n = 16384
1. Tricks and limits
   - Quantization
   - Pruning
   - Early Exits

2. More maths

3. AIPowerMeter

4. Tiny ML

5. Conclusion and Perspectives
Early Exits - motivation

Figure 6: Easy and hard image from [Huang et al., 2017], Copyright Pixel Addict and Doyle
Early Exits - principle

Convolution Block → Convolution Block → Convolution Block

Early Exit

{ dog, cat, tiger, horse, chimp }

Early Exit

{ fish, cat, horse, dog }

Final Exit

{ cat, dog }

Joint work with Yanis Chaigneau, Matthieu François, Paul Gay, Simon Lebeaud, Jordy Palafox, Fatou Kiné Sow et Nicolas Tirel.

Vers un apprentissage profondément plus économique
Early Exists - Measure
Early Exists - Yolo v5

Vers un apprentissage profondément plus économique

joint work with Yanis Chaigneau, Matthieu François, Paul Gay, Simon Lebeaud, Jordy Palafox, Fatou Kiné Sow et Nicolas
- BNN: real gain with Larq (see details here)
- Pruning: real gain with block-sparsity (see details here)
- Early Exits: some gain observed here, nice for model parallelism (work in progress).
Tricks and limits - Conclusion

- Difficult to observe real gain,
- Hardware dependent, model parallelism,
- SGD dependent.
1. Tricks and limits

2. More maths

3. AIPowerMeter

4. Tiny ML

5. Conclusion and Perspectives
Deep Learning is an optimization problem

- Optimization problem to deal with:

\[
\min_{W=(W_1,...,W_L) \in \mathbb{R}^p} \sum_{i=1}^{n} \ell(g_W(X_i), Y_i),
\]

where the number of parameters \( p \) is huge and \( n \) is big.

- Solved by Stochastic Gradient descent:

\[
W^{(t+1)} = W^{(t)} - \alpha \nabla_W \left( \sum_{i=1}^{n} \ell(g_W(X_i), Y_i) \right) [W^{(t)}]
\]
A convex function is dually defined as:

\[ f(y) \geq f(x) + \nabla f(x) \cdot (y - x), \forall x, y. \]

For \( y = \arg \min f(x) \), we have:

\[ -\nabla f(x) \cdot (y - x) \geq 0. \]
SGD - Origin

A convex function is dually defined as:

\[ f(y) \geq f(x) + \nabla f(x) \cdot (y - x), \forall x, y. \]

For \( y = \text{arg min } f(x) \), we have:

\[ -\nabla f(x) \cdot (y - x) \geq 0. \]

WARNING : No constraint about energy consumption
Gradient to Mirror descent

Gradient descent can be written as:

\[
W^{(t+1)} := \arg \min_{W \in \mathbb{R}^p} \left\{ \eta \nabla f(W^{(t)}) \cdot W + \frac{\|W - W^{(t)}\|^2}{2} \right\}.
\]
Gradient to Mirror descent

Gradient descent can be written as:

\[ W^{(t+1)} := \arg \min_{W \in \mathbb{R}^p} \left\{ \eta \nabla f(W^{(t)}) \cdot W + \frac{\|W - W^{(t)}\|^2}{2} \right\} . \]

⇒ no localization and pure Euclidean setting
Gradient to Mirror descent

Mirror descent solves:

\[ W^{(t+1)} := \arg \min_W \left\{ \eta \nabla f(W^{(t)}) \cdot W + B_\Phi(W, W^{(t)}) \right\}, \]

where \( B_\Phi(W, W^{(t)}) = \Phi(W) - \Phi(W^{(t)}) - \nabla \Phi(W^{(t)}) \cdot (W - W^{(t)}) \)

is a Bregman divergence.
Gradient to Mirror descent

Mirror descent solves:

$$W^{(t+1)} := \arg \min_W \left\{ \eta \nabla f(W^{(t)}) \cdot W + B_\Phi(W, W^{(t)}) \right\},$$

where $B_\Phi(W, W^{(t)}) = \Phi(W) - \Phi(W^{(t)}) - \nabla \Phi(W^{(t)}) \cdot (W - W^{(t)})$ is a Bregman divergence.

- For $\Phi(W) = \frac{\|W\|^2}{2}$, mirror descent $\Leftrightarrow$ gradient descent,
Gradient to Mirror descent

Mirror descent solves:

\[ W^{(t+1)} := \arg \min_{W} \left\{ \eta \nabla f(W^{(t)}) \cdot W + B_{\Phi}(W, W^{(t)}) \right\}, \]

where \( B_{\Phi}(W, W^{(t)}) = \Phi(W) - \Phi(W^{(t)}) - \nabla \Phi(W^{(t)}) \cdot (W - W^{(t)}) \)

is a Bregman divergence.

- For \( \Phi(W) = \frac{\|W\|^2}{2} \), mirror descent \( \Leftrightarrow \) gradient descent,
- \( B_{\Phi}(W, W^{(t)}) = \|W - W^{(t)}\|^2 \frac{\nabla^2 \Phi(\omega_t)}{2} \) by Taylor approximation,
Gradient to Mirror descent

Mirror descent solves:

\[ W^{(t+1)} := \arg \min_W \left\{ \eta \nabla f(W^{(t)}) \cdot W + B_\Phi(W, W^{(t)}) \right\}, \]

where \( B_\Phi(W, W^{(t)}) = \Phi(W) - \Phi(W^{(t)}) - \nabla \Phi(W^{(t)}) \cdot (W - W^{(t)}) \) is a Bregman divergence.

- For \( \Phi(W) = \frac{\|W\|^2}{2} \), mirror descent ⇔ gradient descent,
- \( B_\Phi(W, W^{(t)}) = \|W - W^{(t)}\|^2 \nabla^2 \Phi(\omega_t) \) by Taylor approximation,
Sparsity induced NNs

We are looking for a distribution $\rho$ solution of:

$$\min_{\rho} \sum_{i=1}^{n} \mathbb{E}_{W \sim \rho} \ell(g_{W}(X_{i}), Y_{i}).$$

- For an entropy potential $\Phi(\rho) = \int \rho \log \rho$, we have $\mathcal{B}_{\Phi}(\rho, \pi) = \mathcal{K}(\rho, \pi)$,
- By chosen $\rho^{(n)}$ a sparsity prior, we get the following risk bound:

$$\sum_{i=1}^{n} \mathbb{E}_{W \sim \rho^{(n)}} \ell(g_{W}(X_{i}), Y_{i}) \leq \inf_{W \in \mathbb{R}^{p}} \left\{ \sum_{i=1}^{n} \ell(g_{W}(X_{i}), Y_{i}) + \alpha \|W\|_{0} \right\}.$$
Resistence to pruning on CIFAR-10

- CNN with 60,000 params,
- SGD with batch size 256 and no acceleration,
- MCMC with 200 iterations by epoch.

joint work with Yanis Chaigneau, Matthieu François, Paul Gay, Simon Lebeaud, Jordy Palafox, Fatou Kiné Sow et Nicolas Tirel
Resistence to pruning on CIFAR-10

- CNN with 60,000 params,
- SGD with batch size 256 and no acceleration,
- MCMC with 200 iterations by epoch.
Greedy (RJ)-MCMC algorithm

Initialization: \( w_1 \sim \pi \). Parameter \( \lambda > 0 \).

For \( m = 1, \ldots, M \) do
  For \( k = 1, \ldots, N \) do
    • Pick a layer \( \ell \in \{1, \ldots, L\} \) at random,
    • Propose \( \tilde{w} \sim p(\cdot|w_k) \),
    • Accept \( w_{k+1} = \tilde{w} \) with proba:
      \[
      \rho = \frac{\exp\{-\lambda \sum_{t \in I_m} \ell(y_t, g_{\tilde{w}}(x_t))\}}{\exp\{-\lambda \sum_{t \in I_m} \ell(y_t, g_{w_k}(x_t))\}} \frac{\pi(\tilde{w})}{\pi(w_k)}.
      \]
With another mathematical framework, we get a simple Accept/Reject optimizer with several nice conclusion:

- (block)-sparsity induced NNs is easy,
- hybrid optimization is possible,
- model parallelism is easier,
- fast training $\iff$ fast inference.
1 Tricks and limits

2 More maths

3 AIPowerMeter

4 Tiny ML

5 Conclusion and Perspectives
"You can’t improve what you don’t measure"

- **What is energy?** Quantitative property that is transferred to a physical system (through work, heat or light).
- **Measured in Joules:** $1 J \rightarrow$ The energy dissipated as heat when an electric current of one ampere passes through a resistance of one ohm for one second.
- **What is power?** → "The amount of energy transferred or converted per unit time" → $J/s = W$
- **kWh?** → One kW sustained for one hour → $1 kW h = 3.6 MJ$
Energy consumption monitoring in IT

What to monitor?

- **Data Centers**
  Anne Cécile Organo et al
  Papli, Likwid

- **One Particular Hardware**
  Omegawatt (Inria)

- **Deep Learning Applications**
  AI-PowerMeter, CodeCarbon

- **Fine grained measurement**
  Omegawatt (Inria)

Needs

granularity

- High frequency monitors
- One outlet with power meters
- Compute
- Voltage
- Low frequency measurement

joint work with Yanis Chaigneau, Matthieu François, Paul Gay, Simon Lebeaud, Jordy Palafox, Fatou Kiné Sow et Nicolas Tirel

Vers un apprentissage profondément plus économique
Energy consumption monitoring in IT

What to monitor?

Data Centers
Anne Cécile Orgerie et al
Papi, Likwid

One Particular Hardware
Omegawatt (Inria)

Deep Learning Applications
AIPowerMeter, CodoCarbon

Fine grained measurement
Omegawatt (Inria)

Needs

Joint work with Yanis Chaigneau, Matthieu François, Paul Gay, Simon Lebeaud, Jordy Palafox, Fatou Kiné Sow et Nicolas Tirel

Vers un apprentissage profondément plus économique
CPU

- Instructions set: boolean, floating operations
- Registers: fast memory used by the ALU (10-100 registers with 8-64 bits)
- Memory: Closer to the CPU → smaller and faster
- Moving data up and down the memory hierarchy costs time and power → optimizations
GPU

- GPU: Consumes more than the whole computer!
- Thousands of cores to enable parallelism
- Higher latency, Higher memory throughput

More power hungry and requires a CPU → BUT Energy efficient since the computations is faster
Monitoring the energy consumption of a Deep Learning algorithm

Figure 7: Different manners to monitor each components
RAPL for CPU energy consumption monitoring

- Running Average Power Limit (Sandy bridge architecture in 2011)
- Reports the accumulated energy consumption recording at 1000Hz
- Monitor the energy consumption of different parts: CPU, RAM, System on Chip energy consumption, Processor graphics on the socket.
- Command:
  
  ```
  sudo chmod -R 755 /sys/class/powercap/intel-rapl/
  ```

**Figure 8:** K. N. Khan et al. 2018
NVIDIA-SMI to monitor GPU consumption

- NVIDIA System Management Interface (CUDA)
- +/- 5% accuracy of current power draw. Memory usage per gpu and per process
- Command: `nvidia-smi -q -x`

Figure 9: K. N. Khan et al. 2018
In practice: AI PowerMeter

- Developed by GreenAI UPPA, to measure the efficiency of your deep learning recording CPU and GPU
- By and for data scientists
- It uses Nvidia-smi and RAPL as well as psUtil (to compute by program)

https://github.com/GreenAI-Uppa/AIPowerMeter
Figure 10: Wattmeter is more precise, but AiPowermeter permits to monitor the evolution of the power supply.
Application

- Example of the training of AlexNet for image classification.
- Energy consumed during training: $672kJ = 0.187kWh$ ($\sim 11gCO2eq$)
• In the lab, we are interested in embedded systems: Rasberrypi, Jetson Cards or micro-controllers
• Constraints exist to monitor (GPU, RAPL not available for ARM processors, Nvidia-SMI not working on Jetson...)

Figure 12: Find the better compromise between power and memory!
1. Tricks and limits
2. More maths
3. AIPowerMeter
4. Tiny ML
5. Conclusion and Perspectives
**Figure 13**: Benchmark of the tiny ML field [NEUTON.AI, 2022]

- **Edge ML**: 1 MB
- **Tiny ML**: 100 KB
- **Ultra-Tiny ML**: 30 KB
- **New opportunities!**: 10 KB

96% of today's cases involve 1 MB.
4% of cases involve 100 KB.
New opportunities involve 30 KB.

**Tiny ML**
What is Tiny ML?

- What is tinyML?
  
  *TinyML: When a neural network model can be run at an energy cost of below 1 mW*

---

**Figure 14:** Definition of TinyML by Pete Warden [NEUTON.AI, 2022]
Tiny ML?

- This presentation is highly inspired by the book *TinyML: Machine Learning with TensorFlow Lite on Arduino and Ultra-Low-Power Microcontrollers* by Warden and Situnayake [Pete Warden, 2019].

![Figure 15: Different models for different capacities](image)

It is also inspired by many presentations yielded at the TinyML Summit every year (March 2022)
Why?

- Function – wanting a smart device to act quickly and locally (independent of the Internet).
- Cost – accomplishing this with simple, lower cost hardware.
- Privacy – not wanting to share all sensor data externally.
- Efficiency – smaller device form-factor, energy-harvesting or longer battery life.
Limitations in terms of hardware

- Decrease in energy consumption $\rightarrow$ limitations in sRAM memory, flash memory, microprocessor capacities

![Figure 16: Per-block memory usage of MobileNetV2 [Ji Lin, 2021]](image)

$\rightarrow$ What is important is the Peak memory!
A typical microcontroller system consists of a processor core, an on-chip SRAM block and an on-chip embedded flash.

Constraints

- Peak memory usage of the model computations < memory usage.
- Number of parameters in the model < flash memory storage
- Model size and the peak memory < 250 KB each;
- CNN computation < 60 million multiply-adds per inference at high accuracy.
Comparison between hardwares

<table>
<thead>
<tr>
<th>Micro-controller</th>
<th>Price</th>
<th>Memory</th>
<th>Specificities</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino Nano 33 BLE Sense</td>
<td>29,70€</td>
<td>256 kB</td>
<td></td>
</tr>
<tr>
<td>SparkFun Edge</td>
<td>$16.50</td>
<td>384kB</td>
<td></td>
</tr>
<tr>
<td>ST Microelectronics STM32F746G Discovery kit</td>
<td>$54.0</td>
<td>340 kB</td>
<td>Screen / included camera</td>
</tr>
</tbody>
</table>

Table 1: Main micro-controllers on the market for tinyML
## Arduino Nano 33 BLE Sense: Components

<table>
<thead>
<tr>
<th>Sensor</th>
<th>Power</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMU</td>
<td>1mW</td>
</tr>
<tr>
<td>Weather (humidity, and temperature)</td>
<td>5µW</td>
</tr>
<tr>
<td>barometric sensor</td>
<td>10 µW</td>
</tr>
<tr>
<td>microphone</td>
<td>300 µW</td>
</tr>
<tr>
<td>Gesture, proximity, light</td>
<td>?</td>
</tr>
<tr>
<td>Bluetooth® Low Energy connectivity</td>
<td>40 mW</td>
</tr>
</tbody>
</table>

**Table 2: Components integrated**

Possibility to connect many sensors such as cameras for recognition (1 mW at 30 FPS for 320 × 320-pixel monochrome image sensor).
Applications of tinyML

Figure 18: Different possible applications of tinyML [Fournarakis, 2021b]
More precision: Patch-Based Learning

Figure 19: Apply filters on patch only: Patch-Based Learning [Ji Lin, 2021]

Figure 20: Reduce the peak memory! Tradeoff between overall computation time and performances
1 Tricks and limits
2 More maths
3 AIPowerMeter
4 Tiny ML
5 Conclusion and Perspectives
Conclusion

- Using actual pipelines needs tricks and engineering,
- Using more maths can lead to new more sustainable optimizers,
- Model parallelism and hardware is very important -> Jordy et Matthieu
References I

Deep rewiring: Training very sparse deep networks.
In International Conference on Learning Representations.

Binaryconnect: Training deep neural networks with binary weights during propagations.
In Advances in neural information processing systems, pages 3123–3131.

Progressive skeletonization: Trimming more fat from a network at initialization.

A practical guide to neural network quantization.
A practical guide to neural network quantization.

[Han, 2021] Han, S. (2021).
Putting ai on a diet: Tinynml and efficient deep learning.


Multi-scale dense networks for resource efficient image classification.


Skeletonization: A technique for trimming the fat from a network via relevance assessment.

A novel approach to building exceptionally tiny models without loss of accuracy.

Tinyml: Machine learning with tensorflow lite on arduino and ultra-low-power microcontrollers.
A comprehensive review of binary neural network.

and Zou, Y. (2016).
Dorefa-net: Training low bitwidth convolutional neural networks
with low bitwidth gradients.