Manage your neural network energy budget with LaNMax !

CMC Microsystems Workshop @Montréal

Accelerating AI – Challenges and Opportunities in Cloud and Edge Computing

Sébastien Henwood, François Leduc-Primeau, Yvon Savaria

POLYTECHNIQUE MONTRÉAL
UNIVERSITÉ D'INGÉNIERIE

March 6, 2020

Contact: `firstname.lastname@polymtl.ca`

This work is supported by IVADO and ReSMiQ

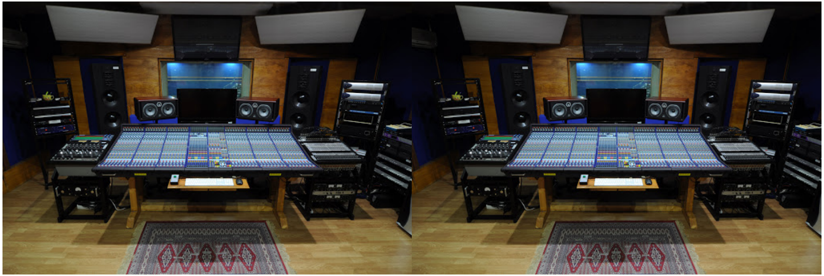
Outline

Outline

Parametric ML 101

Figure 1: Knobs

- Learning = finding the right knobs settings
- "Regular ML" with thousands parameters



(a) Layer 1 (b) Layer 2

Figure 2: Knobs everywhere

- Learning = finding the right knobs settings
- Millions, billions of parameters (NLP mostly)

Storing the parameters

As is, millions of parameters would be quite the burden on any hardware. We have leeway to reduce this burden through e.g.:

- Less degrees of freedom per parameter : quantization (well studied)
- Clever network designs (tricky !)
- Pruning (care of sparse network, specific hardware is needed for full advantage)
- Lately Hacene et al. (2019); Hirtzlin et al. (2019) were reducing supply voltage to the memory with interesting results

Storing the parameters

As is, millions of parameters would be quite the burden on any hardware. We have leeway to reduce this burden through e.g.:

- Less degrees of freedom per parameter : quantization (well studied)
- Clever network designs (tricky !)
- Pruning (care of sparse network, specific hardware is needed for full advantage)
- Lately Hacene et al. (2019); Hirtzlin et al. (2019) were reducing supply voltage to the memory with interesting results

Storing the parameters

As is, millions of parameters would be quite the burden on any hardware. We have leeway to reduce this burden through e.g.:

- Less degrees of freedom per parameter : quantization (well studied)
- Clever network designs (tricky !)
- Pruning (care of sparse network, specific hardware is needed for full advantage)
- Lately Hacene et al. (2019); Hirtzlin et al. (2019) were reducing supply voltage to the memory with interesting results

Storing the parameters

As is, millions of parameters would be quite the burden on any hardware. We have leeway to reduce this burden through e.g.:

- Less degrees of freedom per parameter : quantization (well studied)
- Clever network designs (tricky !)
- Pruning (care of sparse network, specific hardware is needed for full advantage)
- Lately Hacene et al. (2019); Hirtzlin et al. (2019) were reducing supply voltage to the memory with interesting results

On CMOS supply voltage

The dynamic energy consumption formula

$$C \times V^2 = \text{number of parameters} \times \text{number of bits} \times V^2 \times \text{technology dependent constant}$$

- The capacitance C is a constant depending on circuit area
- Static consumption = system online time, proportional to circuit area, i.e. number of parameters
- Dynamic consumption = number of memory accesses. No writes, only reads. Each parameters is read once
- Play with V and achieves quadratic savings !

On CMOS supply voltage

The dynamic energy consumption formula

$$C \times V^2 = \text{number of parameters} \times \text{number of bits} \times V^2 \times \text{technology dependent constant}$$

- The capacitance C is a constant depending on circuit area
- Static consumption = system online time, proportional to circuit area, i.e. number of parameters
- Dynamic consumption = number of memory accesses. No writes, only reads. Each parameters is read once
- Play with V and achieves quadratic savings !

On CMOS supply voltage

The dynamic energy consumption formula

$$C \times V^2 = \text{number of parameters} \times \text{number of bits} \times V^2 \times \text{technology dependent constant}$$

- The capacitance C is a constant depending on circuit area
- Static consumption = system online time, proportional to circuit area, i.e. number of parameters
- Dynamic consumption = number of memory accesses. No writes, only reads. Each parameters is read once
- Play with V and achieves quadratic savings !

On CMOS supply voltage

The dynamic energy consumption formula

$$C \times V^2 = \text{number of parameters} \times \text{number of bits} \times V^2 \times \text{technology dependent constant}$$

- The capacitance C is a constant depending on circuit area
- Static consumption = system online time, proportional to circuit area, i.e. number of parameters
- Dynamic consumption = number of memory accesses. No writes, only reads. Each parameters is read once
- Play with V and achieves quadratic savings !

An optimisation problem

No free lunches, sorry

Reducing voltage as in near threshold CMOS will increase the fault rate p when reading bits (Dreslinski et al. (2010))

$$\eta(p) = -\frac{\log(p)}{a}$$

with η the normalized consumption and a a technology dependent parameter

An optimisation problem

No free lunches, sorry

Reducing voltage as in near threshold CMOS will increase the fault rate p when reading bits (Dreslinski et al. (2010))

$$\eta(p) = -\frac{\log(p)}{a}$$

with η the normalized consumption and a a technology dependent parameter

We seek to jointly optimize an energy-capability trade-off through a fault rate parameter p that (i) degrades the capability when going up and (ii) reduces the energy when going up; while retaining the maximum capability for the lowest energy.
We exploit the known relationship between p and η (and *in fine* V) to measure the trade-off.

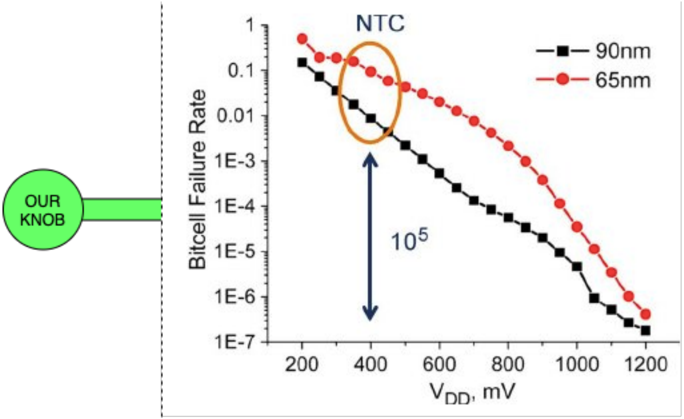
Our work

Hyp.: we can improve the supply voltage idea by letting the network find its own "best" supply voltage

Outline

Problem statement

We want to find *efficiently* the fault rate p that gives the best capability to the neural network for the lowest energy *without additional mechanisms*.



Adapted from Dreslinski et al. (2010)

Efficiently

- Finding the fault rate should have a small training overhead, idealy not at the cost of more training epochs

Implication

Make use of each existing epoch to gain information on the energy-capability trade-off

- **Why ?** Here are the existing training time for large NLP models as reported by Nvidia.

Efficiently

- Finding the fault rate should have a small training overhead, idealy not at the cost of more training epochs

Implication

Make use of each existing epoch to gain information on the energy-capability trade-off

- Why ? Here are the existing training time for large NLP models as reported by Nvidia.

Efficiently

- Finding the fault rate should have a small training overhead, idealy not at the cost of more training epochs

Implication

Make use of each existing epoch to gain information on the energy-capability trade-off

- **Why ?** Here are the existing training time for large NLP models as reported by Nvidia.

BERT-Large Training Times on GPUs

Time	System	Number of Nodes	Number of V100 GPUs
47 min	DGX SuperPOD	92 x DGX-2H	1,472
67 min	DGX SuperPOD	64 x DGX-2H	1,024
236 min	DGX SuperPOD	16 x DGX-2H	256

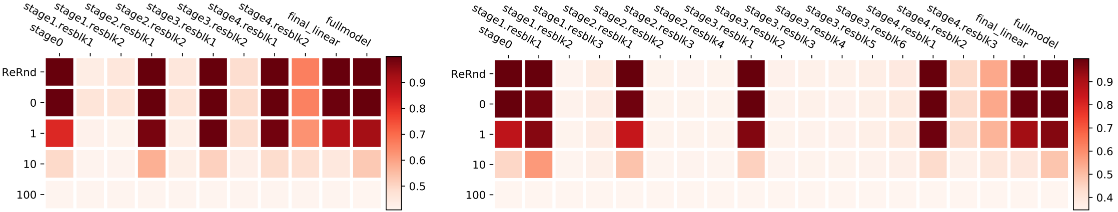
Without additional mechanism

Explored in Hacene et al. (2019) is the use of circuit-level error detection which zero-out faulty weights. This kind of Error-Correction mechanism is known and usable !
But this adds up hardware complexity.

Implication

No ECC or such mechanism, the bits are used by the network as they are read.

With a small training overhead constraint, we could easily add degrees of freedom to the storage energy optimisation : e.g. a fault rate per layer of the neural network.
This would concur with Zhang et al. (2019) (see below): layers don't share the same sensitivity to randomness.



Imagenet trained ResNet 18 and 50 layerwise sensibility to weights rewinding

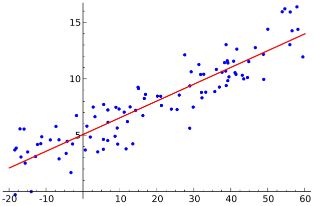
How-to

Usually, the fault rate is an hyperparameter : we propose to learn it with numerical gradients on a per-epoch basis.

Numerical gradient components

$$\nabla_p = \mathrm{OLS}_p(\{\alpha \sum_\ell E_{i,\ell} + \mathrm{loss}_i \,|\forall i \in \text{mini-batch}\}) \text{ coefficients}$$

Moreover, the trade-off is controlled by a parameter α: the bigger α the more emphasis on energy reduction there will be.



Layerwise Noise Maximisation

Algorithm overview

1. Initialize a fault rate p_ℓ per ℓ layer of the network
2. Begin an SGD like-optimization algorithm, for all epochs
    1. Sample neural network's weights at the current fault rate + some randomness
    2. Forward the current mini-batch
    3. Store the current energy-capability trade-off
    4. Do the usual backpropagation and gradient descent
    5. When all mini-batches have been done : linear regression on the trade-off points and numerical gradient descent
    6. Next epoch
3. Return the neural network weights and the layerwise fault rate

Outline

An optimisation scenario

- Moons et al. (2018) proposed that quantized network may be optimal in the energy-capability trade-off. Thus we test our method on a 1-bit weighted Wide Residual Network (Zagoruyko and Komodakis (2016)) that has shown good results in previous works.
- The net is binarized with Binary Connect Courbariaux et al. (2015) on the Conv. and FC layers. These layers have the additional trainable fault rate !
- The faults are uniformly drawn at the rate p each time a *forward* pass is done.
- We use CIFAR-10 dataset with Adam and standard hyperparameters.

An optimisation scenario

- Moons et al. (2018) proposed that quantized network may be optimal in the energy-capability trade-off. Thus we test our method on a 1-bit weighted Wide Residual Network (Zagoruyko and Komodakis (2016)) that has shown good results in previous works.
- The net is binarized with Binary Connect Courbariaux et al. (2015) on the Conv. and FC layers. These layers have the additional trainable fault rate !
- The faults are uniformly drawn at the rate p each time a *forward* pass is done.
- We use CIFAR-10 dataset with Adam and standard hyperparameters.

An optimisation scenario

- Moons et al. (2018) proposed that quantized network may be optimal in the energy-capability trade-off. Thus we test our method on a 1-bit weighted Wide Residual Network (Zagoruyko and Komodakis (2016)) that has shown good results in previous works.
- The net is binarized with Binary Connect Courbariaux et al. (2015) on the Conv. and FC layers. These layers have the additional trainable fault rate !
- The faults are uniformly drawn at the rate p each time a *forward* pass is done.
- We use CIFAR-10 dataset with Adam and standard hyperparameters.

An optimisation scenario

- Moons et al. (2018) proposed that quantized network may be optimal in the energy-capability trade-off. Thus we test our method on a 1-bit weighted Wide Residual Network (Zagoruyko and Komodakis (2016)) that has shown good results in previous works.
- The net is binarized with Binary Connect Courbariaux et al. (2015) on the Conv. and FC layers. These layers have the additional trainable fault rate !
- The faults are uniformly drawn at the rate p each time a *forward* pass is done.
- We use CIFAR-10 dataset with Adam and standard hyperparameters.

Comparing at isoaccuracy

- Can LaNMax provide efficient nets with higher accuracy than reliable smaller networks ?
- As we vary the size of the network, we note ρ the nb. of params. w.r.t. reference network (36 millions parameters).

Comparing at isoaccuracy

- Can LaNMax provide efficient nets with higher accuracy than reliable smaller networks ?
- As we vary the size of the network, we note ρ the nb. of params. w.r.t. reference network (36 millions parameters).

Comparing at isoaccuracy

- Can LaNMax provide efficient nets with higher accuracy than reliable smaller networks ?
- As we vary the size of the network, we note ρ the nb. of params. w.r.t. reference network (36 millions parameters).

A fair comparison

For our results to be relevant, we will only compare networks that achieve the same accuracy.

Beforehand: are All Layers Equals ?

Recall earlier : does our scenario verify the layerwise sensitivity to randomness?
Train with 1% fault rate on global, test with maximum fault rate per layer (i.e. 50%).

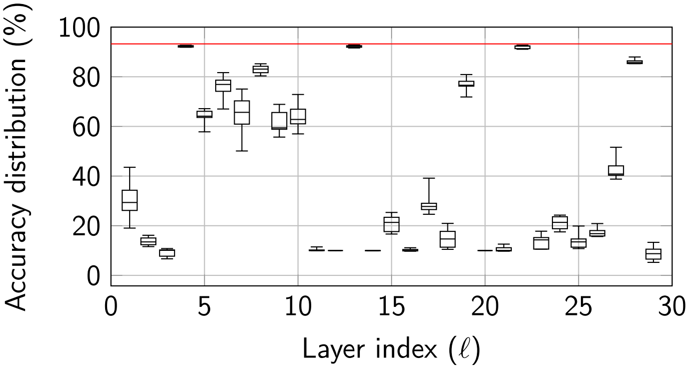
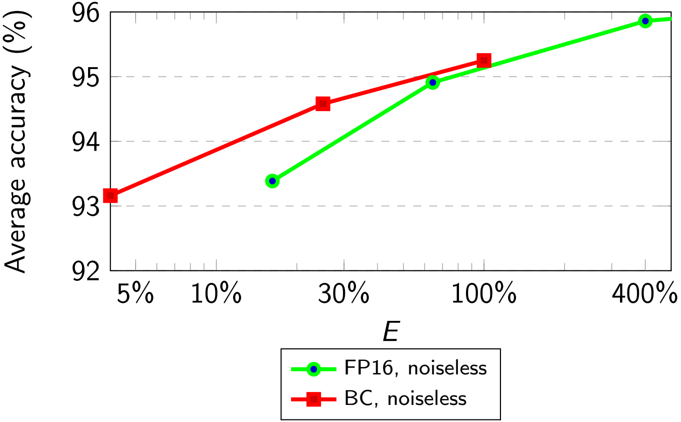


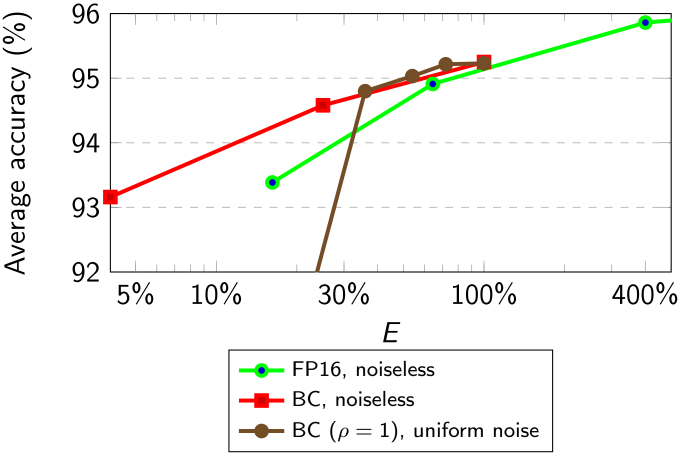
Figure 3: Sensitivity analysis under uniform noise $p = 1\%$

Comparing LaNMax VS net configs. relative energy



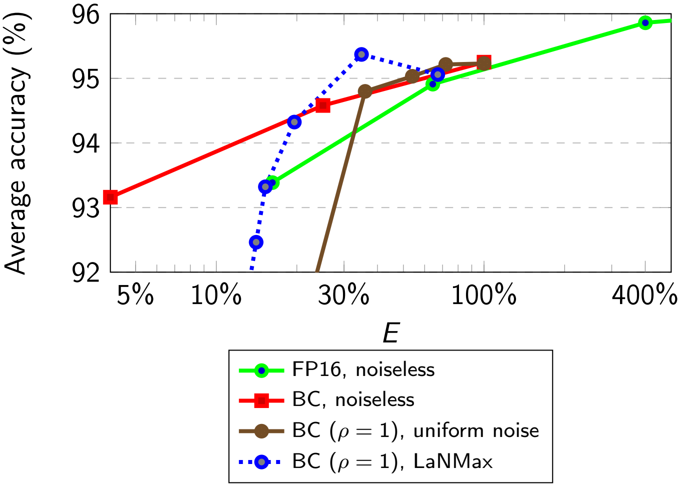
Difference in model size ⇔ difference in energy !

Comparing LaNMax VS net configs. relative energy



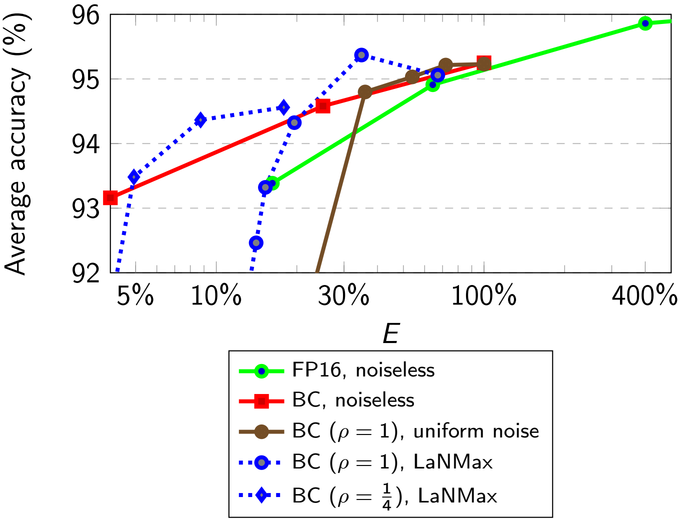
Difference in memory reliability $\Leftrightarrow$ difference in energy !

Comparing LaNMax VS net configs. relative energy

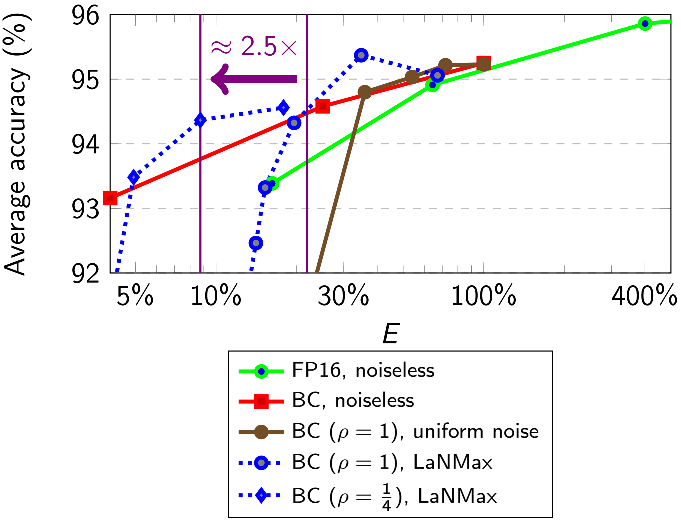


LaNMax optimised memory reliability ⇔ even less energy !

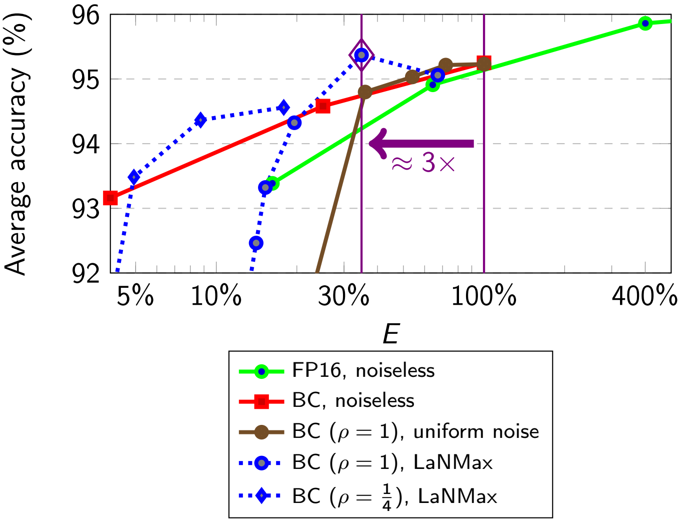
Comparing LaNMax VS net configs. relative energy



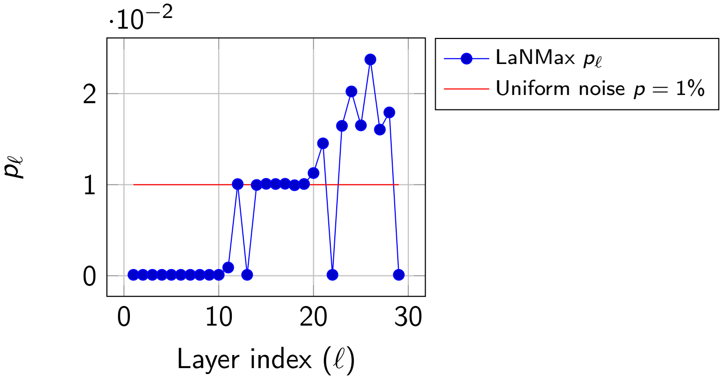
Comparing LaNMax VS net configs. relative energy



Comparing LaNMax VS net configs. relative energy



We can plot the learned fault rate at $\diamond$:



Instead of unreliable small layers from our layerwise sensitivity experiment, we have achieved a fault rate that prioritize unreliability on the largest layers. Good for E !

Outline

Take-away

- Neural nets are terribly good at learning robustness
- Binary neural nets are more robusts than people thoughts, with the adequate tools (not Dropout)
- Exploit this robustness by learning a layerwise fault rate during training
- Deploy a neural network on a server/embedded system at roughly a third of the storage energy cost

Take-away

- Neural nets are terribly good at learning robustness
- Binary neural nets are more robusts than people thoughts, with the adequate tools (not Dropout)
- Exploit this robustness by learning a layerwise fault rate during training
- Deploy a neural network on a server/embedded system at roughly a third of the storage energy cost

Take-away

- Neural nets are terribly good at learning robustness
- Binary neural nets are more robusts than people thoughts, with the adequate tools (not Dropout)
- Exploit this robustness by learning a layerwise fault rate during training
- Deploy a neural network on a server/embedded system at roughly a third of the storage energy cost

Take-away

- Neural nets are terribly good at learning robustness
- Binary neural nets are more robusts than people thoughts, with the adequate tools (not Dropout)
- Exploit this robustness by learning a layerwise fault rate during training
- Deploy a neural network on a server/embedded system at roughly a third of the storage energy cost

This work will be presented at AICAS 2020 and is accessible on Arxiv for details (*arXiv:1912.10764 [cs.LG]*).

Bib.

M. Courbariaux, Y. Bengio, and J. P. David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in Neural Information Processing Systems*, pages 3123–3131, 2015. ISBN 1754-5692. doi: arXiv:1412.7024.

R. G. Dreslinski, M. Wieckowski, D. Blaauw, D. Sylvester, and T. Mudge. Near-Threshold Computing: Reclaiming Moore's Law Through Energy Efficient Integrated Circuits. *Proceedings of the IEEE*, 98(2):253–266, 2 2010. ISSN 0018-9219. doi: 10.1109/JPROC.2009.2034764.

G. B. Hacene, F. Leduc-Primeau, A. B. Soussia, V. Gripon, and F. Gagnon. Training modern deep neural networks for memory-fault robustness. In *Proceedings - IEEE International Symposium on Circuits and Systems*, volume 2019-May, 2019. ISBN 9781728103976. doi: 10.1109/ISCAS.2019.8702382.

T. Hirtzlin, M. Bocquet, J.-O. Klein, E. Nowak, E. Vianello, J.-M. Portal, and D. Querlioz. Outstanding Bit Error Tolerance of Resistive RAM-Based Binarized Neural Networks. In *2019 IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, pages 288–292, 2019. ISBN 978-1-5386-7884-8. doi: 10.1109/AICAS.2019.8771544.

B. Moons, K. Goetschalckx, N. Van Berckelaer, and M. Verhelst. Minimum energy quantized neural networks. In *Conference Record of 51st Asilomar Conference on Signals, Systems and Computers, ACSSC 2017*, volume 2017-October, pages 1921–1925. Institute of Electrical and Electronics Engineers Inc., 4 2018. ISBN 9781538618233. doi: 10.1109/ACSSC.2017.8335699.

S. Zagoruyko and N. Komodakis. Wide Residual Networks. In *British Machine Vision Conference 2016, BMVC 2016*, pages 1–87, 2016. doi: 10.5244/C.30.87.

C. Zhang, S. Bengio, and Y. Singer. Are All Layers Created Equal? *arXiv preprint arXiv:1902.01996*, 2 2019.