Neural Network Precision Tuning Using Stochastic Arithmetic

Quentin Ferro Stef Graillat Thibault Hilaire Fabienne Jézéquel
Basile Lewandowski

LIP6/PEQUAN, Sorbonne Université, CNRS, France

31 March - 1 April 2022

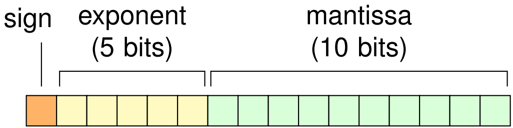
Table of contents

Table of Contents

Introduction

IEEE754 Standard types

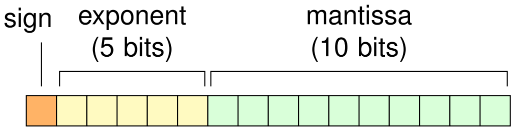
Format	Name	Length	Sign	Mantissa Length	Exponent Length
binary16	Half	16 bits	1 bit	11 bits	5 bits
binary32	Single	32 bits	1 bit	24 bits	8 bits
binary64	Double	64 bits	1 bit	53 bits	11 bits



binary16 format

IEEE754 Standard types

Format	Name	Length	Sign	Mantissa Length	Exponent Length
binary16	Half	16 bits	1 bit	11 bits	5 bits
binary32	Single	32 bits	1 bit	24 bits	8 bits
binary64	Double	64 bits	1 bit	53 bits	11 bits



binary16 format

Reduced precision:

- Shorter execution time ☺
- Less volume of results exchanged (less memory used) ☺
- Less energy consumption ☺
- Less accurate results - rounding errors ☹

Related Work: Precision Auto-Tuning

Static Approach

- FPTuner [Chiang et al., 2017], SALSA [Damouche and Martel, 2018], TAFFO [Cherubin et al., 2020]

Dynamic Approach

- CRAFT HPC [Lam et al., 2013], Precimonious [Rubio-González et al., 2013], HiFPTuner [Guo et al., 2018] and PROMISE [Graillat et al., 2019]

On GPUs

- AMPT-GA [Kotipalli et al., 2019], GPUMixer [Ho et al., 2019], GRAM [Ho et al., 2021]

Related Work: Neural Network

Security and Robustness

[Singh et al., 2018], [Madry et al., 2019], [Lin et al., 2019], [Rakin et al., 2021], [Tjeng et al., 2019]

Impact of rounding errors

[Zombori et al., 2021]
Actual robustness $\neq$ Robustness given by a verifier

Related Work

Neural Network Precision Tuning [Ioualalen and Martel, 2019]

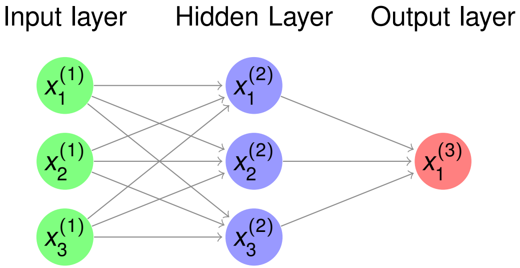
- floating-point auto-tuning algorithm that takes into account a given tolerance
- focus on interpolation networks, i.e. networks computing mathematical function
- precision optimized by solving a linear programming problem

Main differences with our work using PROMISE:

- focus on interpolation networks
- solving a linear programming problem $\neq$ Delta-Debug algorithm
- reference result altered by rounding errors

Neural Network

An artificial neural network is a computing system defined by several neurons distributed on different layers



$$x^{(k+1)} = g^{(k+1)}(W^{(k+1)}x^{(k)} + b^{(k+1)})$$

with W the weight matrix, b the bias vector and g the activation function

Neural Network

The activation function is a non-linear and often monotonous function.

- Sigmoid: $f(x) = \frac{1}{1+e^{-x}}, \forall x \in \mathbb{R}$
- Hyperbolic Tangent ($\texttt{tanh}$)
- Rectified Linear Unit (ReLU): $max(0, x)$
- Softmax: normalizes input vector $x \in \mathbb{R}^n$ into a probability distribution in σ
  $\sigma(x)_j = \frac{e^{x_j}}{\sum_{n=1}^{n} e^{x_i}}, \forall j = 1...n$

Discrete Stochastic Arithmetic (DSA) [Vignes 2004]

Classic arithmetic

$\boxed{A \oplus B \rightarrow R}$

$R = 3.14237654356891$

DSA

Random rounding

$A_1 \oplus B_1 \longrightarrow R_1$

$A_2 \oplus B_2 \longrightarrow R_2$

$A_3 \oplus B_3 \longrightarrow R_3$

$R_1 = \mathbf{3.14}1354786390989$
$R_2 = \mathbf{3.14}3689456834534$
$R_3 = \mathbf{3.14}2579087356598$

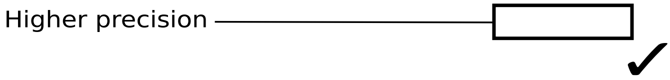
Rounding error analysis based on the CESTAC method:

- Allows the estimation of round-off error propagation
- Based on random rounding mode
- Computing N samples of a result R, value of R becomes $\bar{R}$
  In practice $N = 3$
- Number of correct digits thanks to Student's test with confidence level 95%

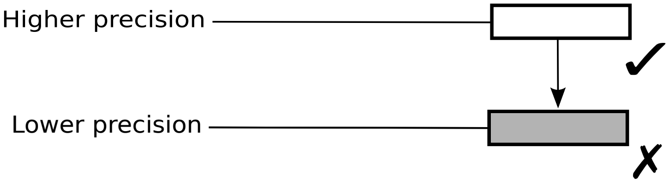
- implements stochastic arithmetic for C/C++ or Fortran codes
- provides stochastic types: 3 values of a variable + 1 integer being the accuracy
- returns value with the exact number of correct digits

PROMISE promise.lip6.fr ([Graillat et al., 2019], [Jézéquel et al., 2021])

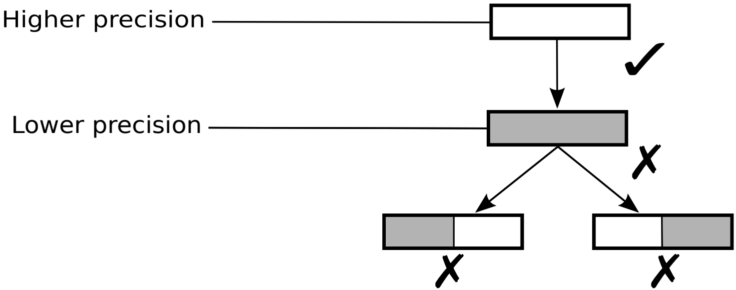
- Provides a mixed-precision code taking in account a required accuracy
- Uses CADNA to validate a configuration
- Uses the Delta-Debug algorithm to test the different configurations, not exhaustive but mean complexity in $O(nlog(n))$ for n variables [Zeller, 2019]


Higher precision
✓

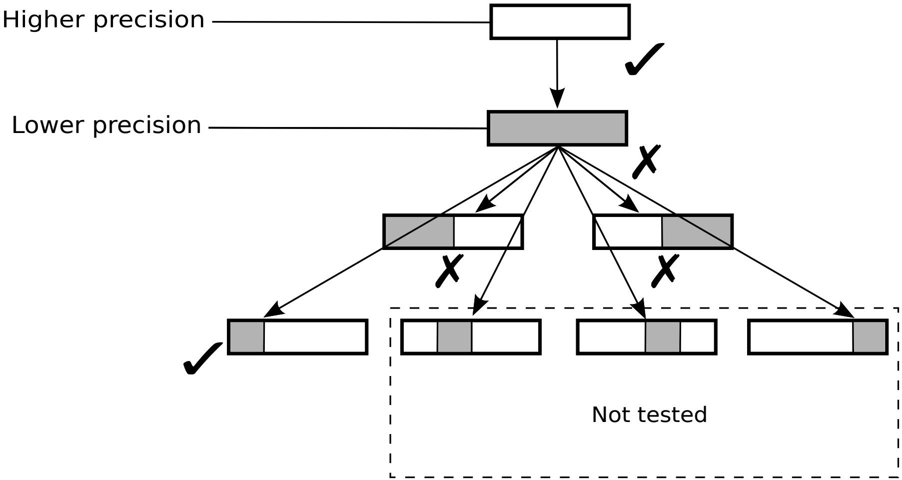
Searching for a valid configuration with 2 types



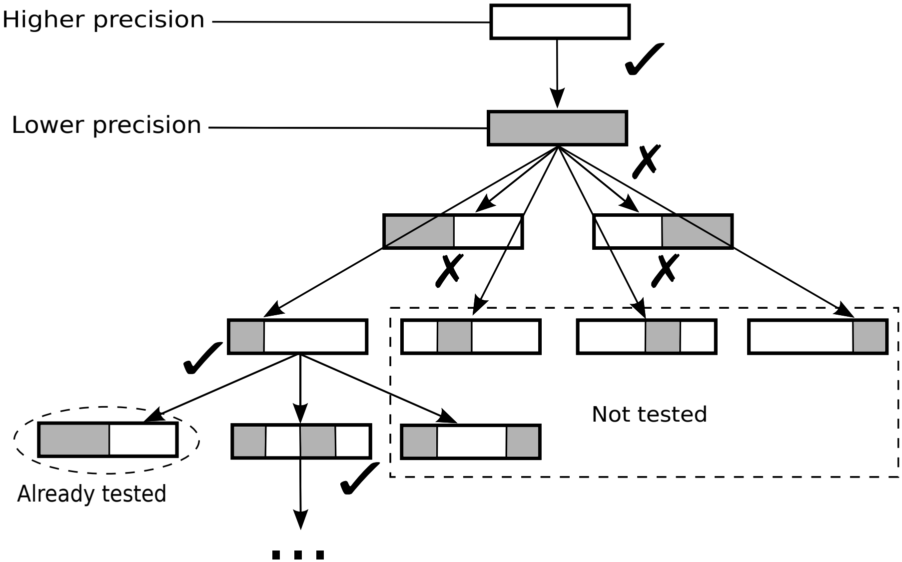
Searching for a valid configuration with 2 types



Searching for a valid configuration with 2 types

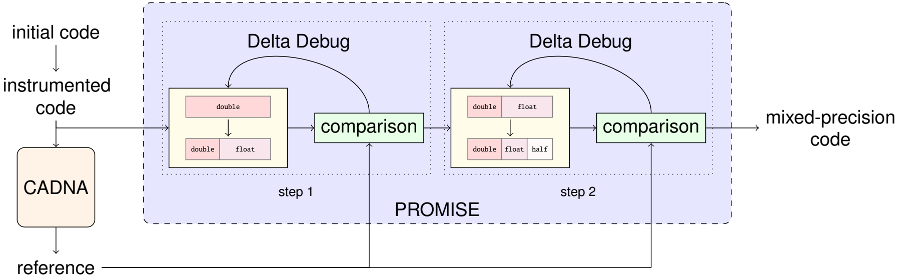


Searching for a valid configuration with 2 types



PROMISE

instrumented code = code with PROMISE variables, custom types variables that PROMISE recognizes and will consider tweaking



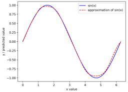
- step 1: lower from double to single precision
- step 2: lower from single to half precision

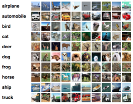
Table of Contents

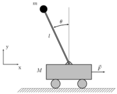
Neural networks studied

4 different Neural Networks:

- Sine NN: approximation of sine function
- MNIST NN: classification of handwritten digits (MNIST Database)
- CIFAR NN: classification of pictures among 10 classes (dogs, cats, deer, car, boat...) (CIFAR10 Database)
- Inverted Pendulum: computation of a Lyapunov function [Chang et al., 2020]







Methodology

- Neural Networks created and trained in Python code with Keras or PyTorch
- Python scripts to pass them into C++ instrumented code

- One type per neuron
- One type per layer
  With input as double precision.

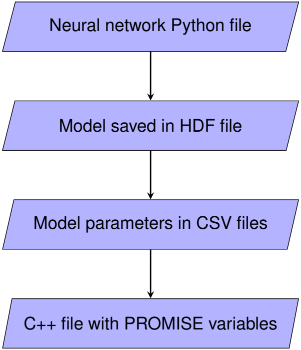


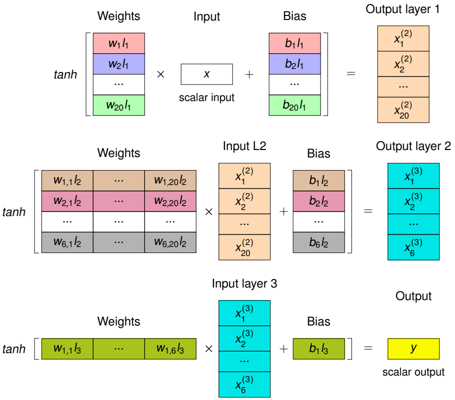
Table of Contents

Sine NN

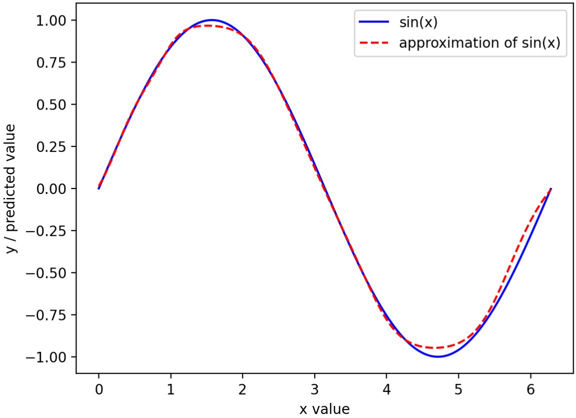
Approximation of sine function:

- Scalar input
- 3 dense layers with tanh activation function:
  - 20 neurons → 21 types to set
  - 6 neurons → 7 types to set
  - 1 neuron → 2 types to set
- Scalar output

➔ 30 types to set in total

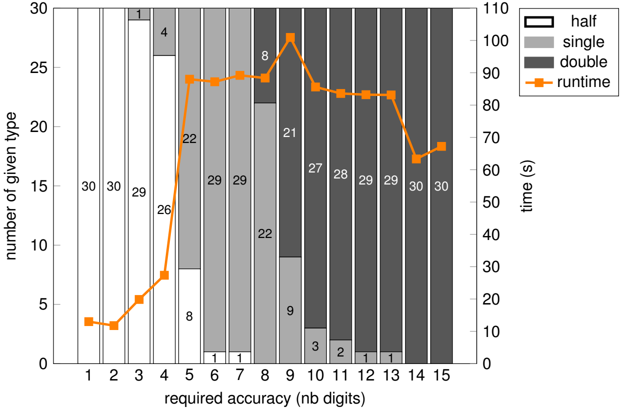


Sine NN

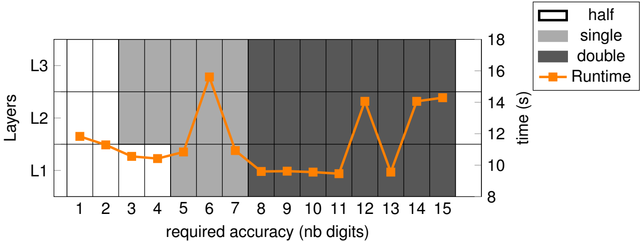


sin(x)
approximation of sin(x)
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
y / predicted value
0
1
2
3
4
5
6
x value

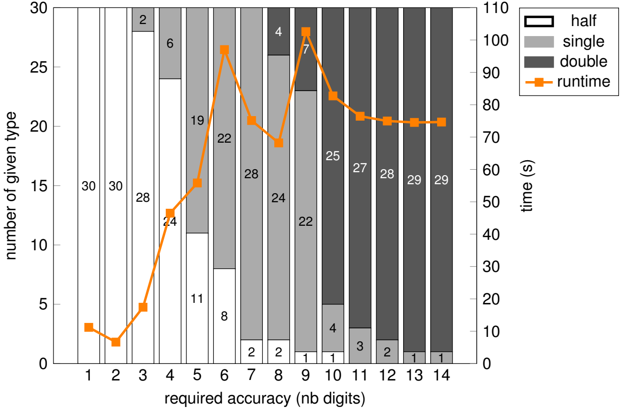
Sine NN w/ input=0.5



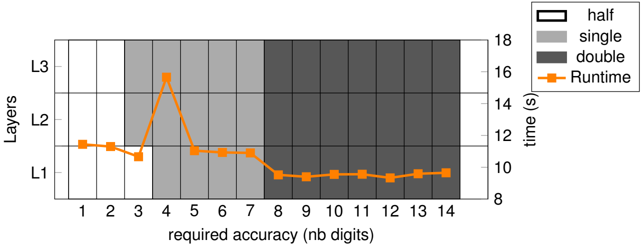
Sine NN w/ input=0.5



Sine NN w/ input=-2.37



Sine NN w/ input=-2.37



MNIST NN

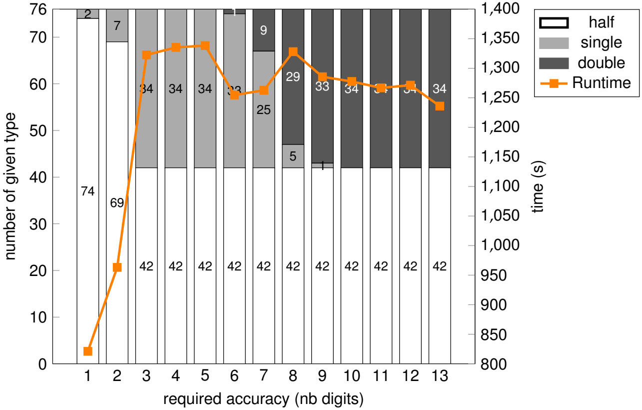
Classification of handwritten digits:

- Input: vector of size 784 (flatten image)
- 2 dense layers:
  - 64 neurons and ReLU activation function $\rightarrow$ 65 types to set
  - 10 neurons and softmax activation function $\rightarrow$ 11 types to set
- output vector of size 10: probability distribution for the 10 different classes

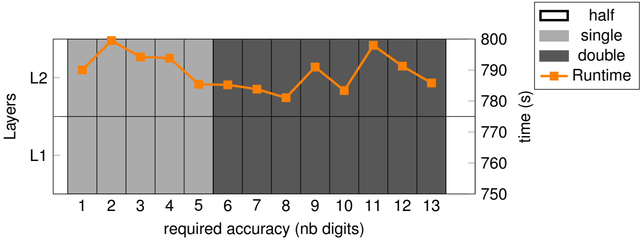
wikipedia.org

➔ 76 types to set in total

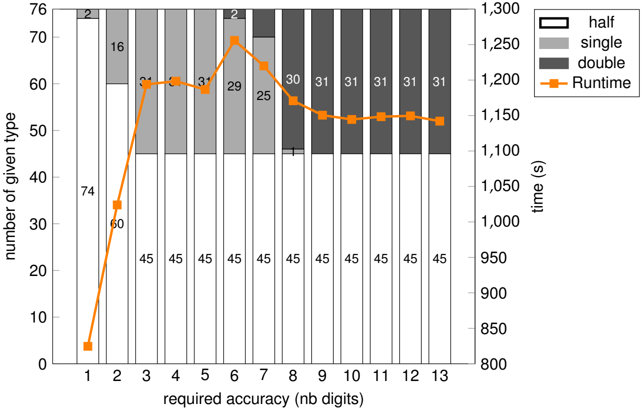
MNIST NN w/ input = test_data[61]



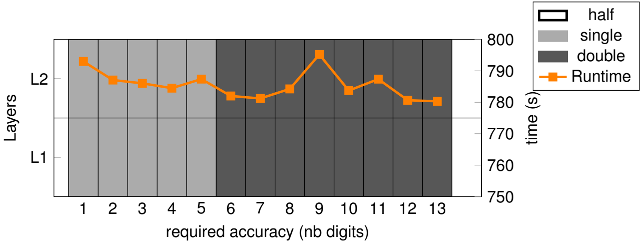
MNIST NN w/ input = test_data[61]



MNIST NN w/ input = test_data[91]



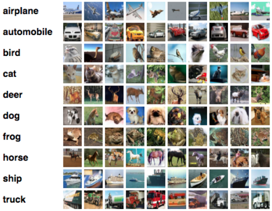
MNIST NN w/ input = test_data[91]



CIFAR NN

Classification of pictures in 10 classes:

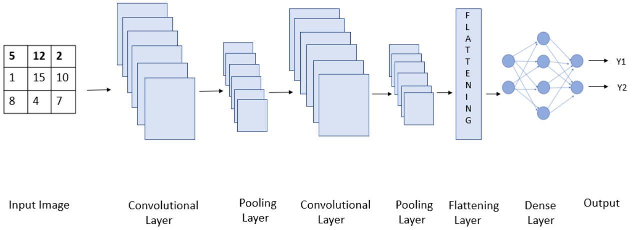
- Input: tensor of shape 32x32x3
- 8 layers:
  - Convolutional layer with 32 neurons and ReLU activation function → 33 types to set
  - Max-pooling layer of size (2x2) → 1 type to set
  - Convolutional layer with 64 neurons and ReLU activation function → 65 types to set
  - Max-pooling layer of size (2x2) → 1 type to set
  - Convolutional layer with 64 neurons and ReLU activation function → 65 types to set
  - Flatten layer → 1 type to set
  - Dense layer of 64 neurons and ReLU activation function → 65 types to set
  - Dense layer of 10 neurons and no activation function → 11 types to set
- output vector of size 10



cs.toronto.edu

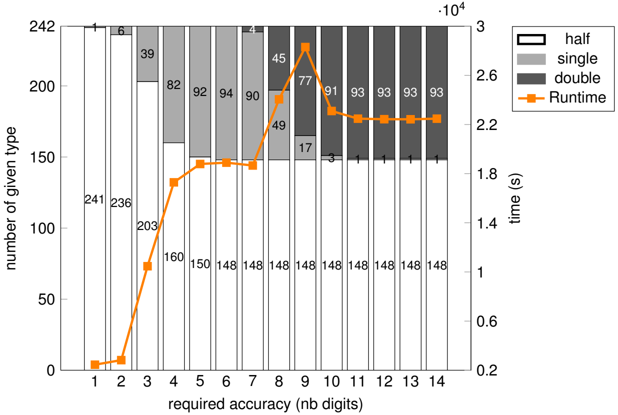
➔ 242 types to set in total

CIFAR NN

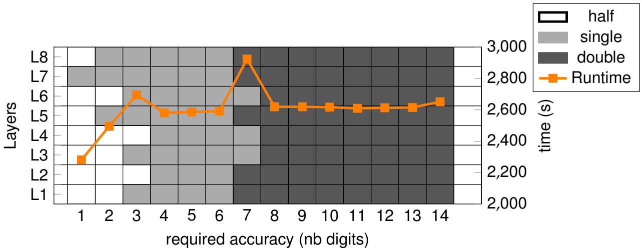


Aarya Brahmane - "Deep Learning with CIFAR-10" - https://towardsdatascience.com/

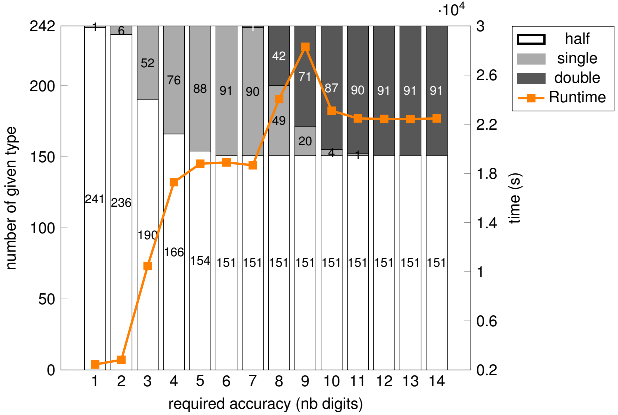
CIFAR NN w/ input=test_data[386]



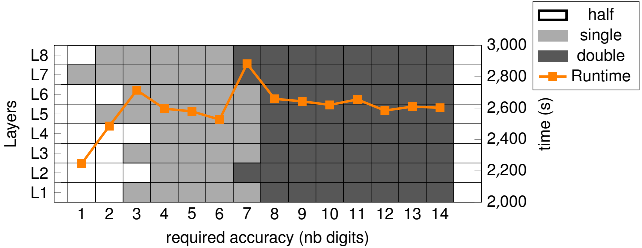
CIFAR NN w/ input=test_data[386]



CIFAR NN w/ input=test_data[731]



CIFAR NN w/ input=test_data[731]

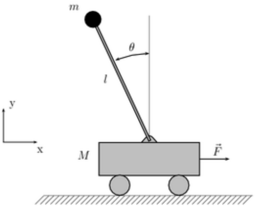


Pendulum NN

Learner to find a Lyapunov function:

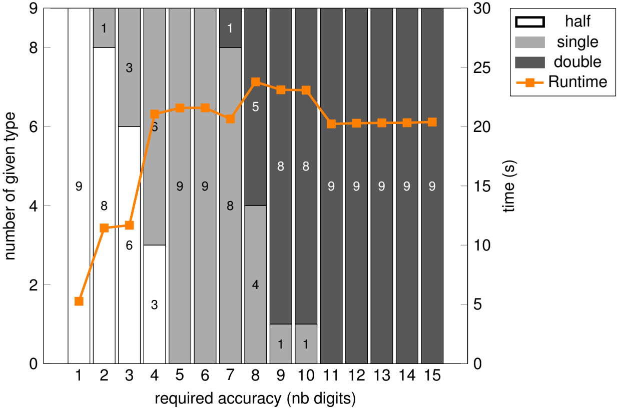
- Input: state vector $x \in \mathbb{R}^2$
- 2 dense layers with tanh activation function:
  - 6 neurons $\rightarrow$ 7 types to set
  - 1 neuron $\rightarrow$ 2 types to set
- output vector of size 10

➔ 9 types to set in total

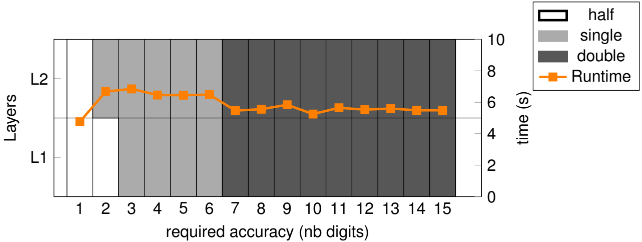


wikipedia.org

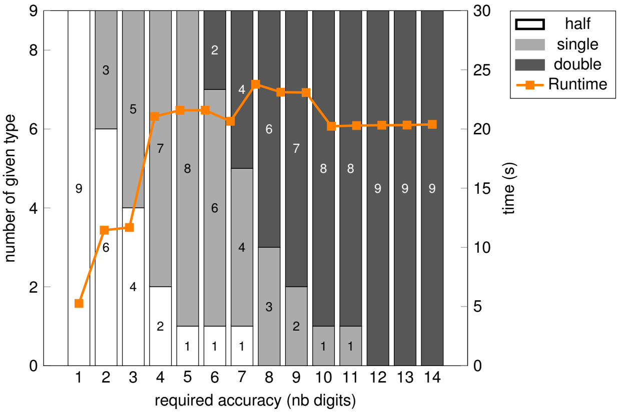
Inverted Pendulum w/ input=(0.5,0.5)



Pendulum NN w/ input=(0.5,0.5)



Inverted Pendulum w/ input=(-3,-6)



Pendulum NN w/ input=(-3,-6)

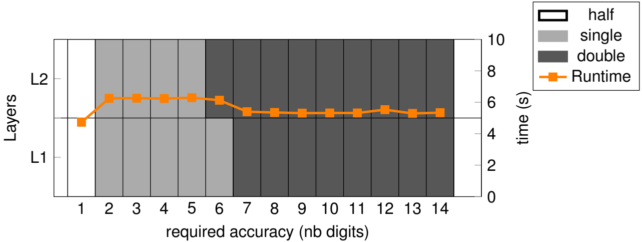


Table of Contents

Conclusion

- Can reduce precision per neuron using PROMISE
- No real advantage considering precision per layer except time cost and starting configuration for tuning per neuron

Future works

- Analyse on more inputs
- Analyse gain in time and memory
- Optimise the Delta-Debug algorithm [Hodován and Kiss, 2016]
- Consider the parallelization of PROMISE
- Extend PROMISE to GPUs and to arbitrary precision on FPGAs
- Use PROMISE with bfloat16

Table of Contents

References I

[Chang et al., 2020] Chang, Y.-C., Roohi, N., and Gao, S. (2020).
Neural Lyapunov Control.
33rd Conference on Neural Information Processing Systems (NeurIPS 2019).
arXiv: 2005.00611.

[Chesneaux, 1995] Chesneaux, J.-M. (1995).
L'arithmétique stochastique et le logiciel CADNA, Habilitation à diriger des recherches.
Université Pierre et Marie Curie, Paris, France.

[Chiang et al., 2017] Chiang, W.-F., Baranowski, M., Briggs, I., Solovyev, A., Gopalakrishnan, G., and Rakamarić, Z. (2017).
Rigorous Floating-Point Mixed-Precision Tuning.
In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages*, POPL 2017, pages 300–315, New York, NY, USA. ACM.

References II

[Damouche and Martel, 2018] Damouche, N. and Martel, M. (2018).
Salsa: An Automatic Tool to Improve the Numerical Accuracy of Programs.
In *6th International Workshop on Automated Formal Methods (AFM)*, volume 5, pages 63–76.

[Graillat et al., 2019] Graillat, S., Jézéquel, F., Picot, R., Févotte, F., and Lathuilière, B. (2019).
Auto-Tuning for Floating-Point Precision with Discrete Stochastic Arithmetic.
Journal of Computational Science, 36:101017.

[Guo and Rubio-González, 2018] Guo, H. and Rubio-González, C. (2018).
Exploiting Community Structure for Floating-Point Precision Tuning.
In *Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis*, pages 333–343, Amsterdam Netherlands. ACM.

References III

[Ho et al., 2021] Ho, N.-M., silva, H. D., and Wong, W.-F. (2021).
GRAM: A Framework for Dynamically Mixing Precisions in GPU Applications.
ACM Transactions on Architecture and Code Optimization, 18(2):1–24.

[Hodován and Kiss, 2016] Hodován, R. and Kiss, A. (2016).
Practical Improvements to the Minimizing Delta Debugging Algorithm:.
In *Proceedings of the 11th International Joint Conference on Software Technologies*, pages 241–248, Lisbon, Portugal. SCITEPRESS - Science and Technology Publications.

[IEEE Computer Society, 2008] IEEE Computer Society (2008).
IEEE Standard for Floating-Point Arithmetic.
IEEE Standard 754-2008.

References IV

[Ioualalen and Martel, 2019] Ioualalen, A. and Martel, M. (2019).
Neural Network Precision Tuning.
In Parker, D. and Wolf, V., editors, *Quantitative Evaluation of Systems*, volume 11785, pages 129–143. Springer International Publishing, Cham.
Series Title: Lecture Notes in Computer Science.

[Jézéquel et al., 2021] Jézéquel, F., Hoseininasab, S. s., and Hilaire, T. (2021).
Numerical Validation of Half Precision Simulations.
In Rocha, Á., Adeli, H., Dzemyda, G., Moreira, F., and Ramalho Correia, A. M., editors, *Trends and Applications in Information Systems and Technologies*, volume 1368, pages 298–307. Springer International Publishing, Cham.
Series Title: Advances in Intelligent Systems and Computing.

References V

[Kotipalli et al., 2019] Kotipalli, P. V., Singh, R., Wood, P., Laguna, I., and Bagchi, S. (2019).
AMPT-GA: Automatic Mixed Precision Floating Point Tuning for GPU Applications.
In *Proceedings of the ACM International Conference on Supercomputing*, pages 160–170, Phoenix Arizona. ACM.

[Laguna et al., 2019] Laguna, I., C. Wood, P., Ranvijay, S., and Bagchi, S. (2019).
GPUMixer: Performance-Driven Floating-Point Tuning for GPU Scientific Applications.
volume 11501, pages 227–246. Springer.
M. Weiland, G. Juckeland, C. Trinitis, and P. Sadayappan.

[Lam et al., 2013] Lam, M. O., Hollingsworth, J. K., de Supinski, B. R., and Legendre, M. P. (2013).
Automatically Adapting Programs for Mixed-Precision Floating-Point Computation.
In *Proceedings of the 27th International ACM Conference on International Conference on Supercomputing*, ICS '13, pages 369–378, New York, NY, USA. ACM.

[Lin et al., 2019] Lin, W., Yang, Z., Chen, X., ZHAO, Q., LI, X., LIU, Z.-l., and HE, J. (2019).
Robustness Verification of Classification Deep Neural Networks via Linear Programming.
Conference on Computer Vision and Pattern Recognition.

[Madry et al., 2019] Madry, A., Makelov, A., Schmidt, L., Tsipras, D., and Vladu, A. (2019).
Towards Deep Learning Models Resistant to Adversarial Attacks.
6th International Conference on Learning Representations, ICLR.
arXiv: 1706.06083.

References VII

[Rakin et al., 2021] Rakin, A. S., Yang, L., Li, J., Yao, F., Chakrabarti, C., Cao, Y., Seo, J.-s., and Fan, D. (2021).
RA-BNN: Constructing Robust & Accurate Binary Neural Network to Simultaneously Defend Adversarial Bit-Flip Attack and Improve Accuracy.
arXiv: 2103.13813 [cs, eess].

[Rubio-González et al., 2013] Rubio-González, C., Nguyen, C., Nguyen, H. D., Demmel, J., Kahan, W., Sen, K., Bailey, D. H., Iancu, C., and Hough, D. (2013).
Precimonious: Tuning Assistant for Floating-Point Precision.
In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC'13, pages 27:1–27:12, New York, NY, USA. ACM.

References VIII

[Singh et al., 2018] Singh, G., Gehr, T., Mirman, M., Püschel, M., and Vechev, M. (2018).
Fast and Effective Robustness Certification.
Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems, NeurIPS, pages 10825–10836.

[Tjeng et al., 2019] Tjeng, V., Xiao, K., and Tedrake, R. (2019).
Evaluating Robustness of Neural Networks with Mixed Integer Programming.
arXiv:1711.07356 [cs].

[Vignes, 2004] Vignes, J. (2004).
Discrete Stochastic Arithmetic for Validating Results of Numerical Software.
Numerical Algorithms, 37(1–4):377–390.

References IX

[Zombori, 2021] Zombori, D. (2021).
Fooling a Complete Neural Network Verifier.
The 9th International Conference on Learning Representations (ICLR).