Optimal Hardware Parameters Prediction for Best Energy-to-Solution of Sparse Matrix Operations Using Machine Learning Techniques

Vojtech Nikl*, Ondrej Vysocky†, Lubomir Riha† and Jan Zapletal†
*IT4Innovations Centre of Excellence, Brno University of Technology, Brno, Czech Republic
inikl@fit.vutbr.cz
†IT4Innovations, VŠB – Technical University of Ostrava, Ostrava-Poruba, Czech Republic
{ondrej.vysocky|lubomir.riha|jan.zapletal}@vsb.cz

***Abstract*—Combinations of 3 hardware parameters (number of threads, core and uncore frequency) were tested for 4 sparse matrix algorithms (matrix-matrix addition, matrix-matrix multiplication and matrix-vector multiplication in 2 formats) on a set of over 2,000 matrices for the purpose of identifying the best energy-to-solution setting for each matrix and sparse matrix operation combination. On this set of data, the possibility of optimal hardware settings prediction based on the properties of each matrix were analysed using neural networks, support vector machines and fast decision tree learners. All 3 classes of algorithms have been proven to be a very effective instrument in a lot of areas including prediction and classification. In neural networks, the input neurons represented properties of a given matrix, output neurons represented the optimal hardware parameters. Network properties (hidden neuron layers, neurons per layer, learning coefficient and training cycles) impact on the prediction accuracy were analysed and the results showed that a network with 30 hidden neurons produced results close to the best achievable. The prediction accuracy of all neural networks ranged from 20–95%, with roughly 70% being the average. Support vector machines were accurate in 60–65% of cases and Fast decision tree learners provided the least accurate predictions, 50–55%.**

***Keywords*—Sparse matrices, neural network, support vector machine, fast decision tree learner, weka, energy efficiency, prediction**

I. Introduction

A sparse matrix is a matrix in which most of the elements are equal to zero. Some common applications are partial differential equations, numerical analysis and linear algebraic operations. Sparse matrices can represent real-world problems ranging from microscopic systems [1] up to whole galaxies [2].

Sparse operations are characterised by low arithmetic intensity, resulting in a challenging memory-bound problem, where underclocking the processor or limiting the number of compute threads can reduce the memory congestion and have a minimal impact on the performance itself.

The goal is to find the optimal hardware setup to save the most energy for each specific set of input parameters. One way is to manually measure all combinations of the hardware parameters and choose the optimal one. However, that quickly becomes inconvenient for increasing number of matrices. This approach is also called Hyper-Parameter Optimisation (HPO).

Bergsta et. al. [3] presented two popular HPO algorithms, neural and deep belief networks with the Gaussian Process approach and the Tree-structured Parzen Estimator approach, and showed that the results obtained from running an image classification problem with 32 hyper-parameters are human and brute force random search competitive, with an average error being just under 15%.

Bergsta et. al. [4] also put an emphasis on the initial hyper-parameter layout. Grid and random layouts were compared on a neural network case study. It showed that random experiments were more efficient than the grid ones for the hyper-parameter optimisation in the case of the most learning algorithms on several data sets. The main reason is that not all hyper-parameters are equally important to tune.

Stamoulis et. al. [5] focused on the hyper-parameter optimisation of neural networks in the direction of power and memory constrains on GTX 1070 and Tegra X1 [6] GPUs from Nvidia. The framework used Bayesian optimisation model and overall the enhancements allowed for up to 57.2× more function evaluations, which yielded significant accuracy improvements by up to 67.6%.

Smithson et. al. [7] showed an approach for reducing the state space of the neural network properties by using another neural network, which reduced this state space by hyper-parameter optimisations. In the end, a Pareto-optimal set of networks was created. Compared to manually designed networks from literature, this technique produced results with nearly identical performance while reducing the associated costs by a factor of 3.

Auto-weka [8] is a JAVA library and a machine learning platform. Auto-sklearn [9] is its sister package written in Python. These frameworks offer a set of popular learners and algorithms for problems where it is hard to identify the best approach. It automatically searches through the joint space of Weka's learning algorithms and their respective hyper-parameter settings to maximise performance, using a state-of-the-art Bayesian optimisation method.

The current state-of-the-art in the hyper-parameters optimisation approaches, outlined in previous paragraphs, show that machine learning techniques, neural networks especially, provide a powerful tool for solving these tasks. This paper, however, presents a unique challenge from the area of sparse matrix operations. Based on the properties of a given ma-

 ISBN: 978-1-61208-655-2

trix, the goal is to successfully predict the optimal hardware parameters in terms of the best energy-to-solution and also the energy consumption itself. This paper focuses mainly on neural networks and their performance tuning as well as comparing them to the support vector machines and fast decision tree learners, provided and recommended by the Weka library for this class of problems.

II. Setup

In this section, all hardware and software used, their versions, settings, properties, algorithms etc. are described. All source codes were compiled with the combination of *Intel Compiler 2017*, `-O3 -xHost` flags and *Intel MKL 2017*to perform the sparse calculations, which were provided as modules on the cluster.

A. Sparse Algorithms

4 sparse algorithms were analysed:

- matrix-matrix addition in CSR format (SpMMadd)
- matrix-matrix multiplication in CSR format (SpMMmult)
- matrix-vector multiplication in CSR format (SpMVmultCSR)
- matrix-vector multiplication in IJV (i.e. COO) format (SpMVmultIJV)

IJV format stores data as a set of coordinates and is more suitable for the matrix-vector multiplication than CSR. The matrix-matrix addition and multiplication adds or multiplies a given matrix with itself (transposed if necessary), respectively, matrix-vector multiplication multiplies a given matrix with its first row as a vector.

B. Matrices

All sparse matrices were sourced from the SuiteSparse Matrix collection [10] [11] in the *Matrix Market File* (.mtx) format. The collection contains 2757 matrices ranging from a single up to almost 2 billion nonzero elements. Due to the resource and allocation limitations of the cluster, only a randomly chosen subset of matrices was used for each sparse operation (1755 for SpMMadd, 1627 for SpMMmult, 2493 for SpMVmultCSR, 2605 for SpMVmultIJV).

C. Neural Network and Weka

The Genann library [12] was used to implement the multilayer fully-connected feedforward neural networks with a sigmoid activation function and the back-propagation learning.

Each network had 226 input neurons representing the parameters of a matrix. The numerical parameters were *Rows*, *Columns*, *Nonzero ratio* and *Symmetry percentage*, each occupying one input neuron. The values of these parameters were normalised to $[0, 1]$ range by the *min-max* normalisation as

$$X_{norm} = \frac{X - X_{min}}{X_{max} - X_{min}} \quad (1)$$

The categorical parameters, normalised using the *one-hot* encoding [13], are *Group* (143 variants), *Kind* (65 variants), Type (12 variants) and *SPD* (symmetric positive definite, 1 variant). Each variant was represented by its own input activating neuron.

Four output neurons represented *Number of threads*, *Core frequency*, *Uncore frequency* (separate L3 cache frequency introduced by Intel in the Haswell architecture) and *Energy*.

Editable analysed parameters are hidden neuron layers (1–5), the number of neurons per hidden layer (10–250), learning coefficients (0.1–0.99), the number of learning cycles (i.e. epochs) (1, 10, 100, 1000) and the training set sizes (10–90% of total data, the rest formed testing data).

For algorithms provided by the Weka library - support vector machines for regression, referred to as *SMOreg*, and fast decision learning trees, referred to as *REPtree*, the csv files with all the input vectors (matrix properties) and experimentally measured data were converted to the *arff* format. The parameters of both of these algorithms were set to default values recommended by the authors.

D. Hardware

All experiments were run on the Taurus supercomputer [14], where each node consists of 2× Intel Xeon E5-2680v3 (12 cores) processors and 64–256 GB RAM. The benchmarked core and uncore frequencies were 1.2, 1.5, 1.8, 2.1 and 2.5 GHz, and additionally 3.0 GHz for the uncore. The numbers of threads tested covered 2, 4, 6, 8 and 12. Ideally, the whole Cartesian product of these parameters should be measured, however, that would inadequatly increase the search space without the benefit of much improved results. The workload was duplicated to both sockets, because sparse routines scale very poorly to a higher number of threads and across NUMA regions and duplicating the calculations gives more accurate energy measurements. The energy consumption difference running the same algorithm and hardware settings among the nodes is roughly ±5% on average.

E. Benchmarking

During the initial experimental data collection phase, each sparse operation for each matrix and each combination of HW settings was run twice to warm up the caches, the measured hot run was repeated to run at least 5 times and for at least 1 second. The energy and time measurement as well as the dynamic hardware parameter settings were done using the *MERIC* library [15] [16], which uses *HDEEM* [17] for high frequency energy measurements. For each sparse matrix, all tests were executed on the same node to reduce the initial I/O overhead, but different matrices run in parallel on randomly allocated nodes.

III. Experimental results

The impact on prediction accuracy was individually evaluated for each dynamic neural network parameter. The accuracy is expressed as the Euclidean norm in a 4-dimensional space of error values as

$$Error\ distance = \sqrt{Err_t^2 + Err_c^2 + Err_u^2 + Err_e^2} \quad (2)$$

 ISBN: 978-1-61208-655-2

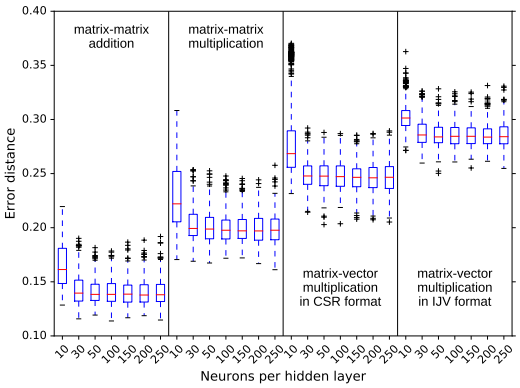


Figure 1: Impact of number of neurons on prediction accuracy of NN.

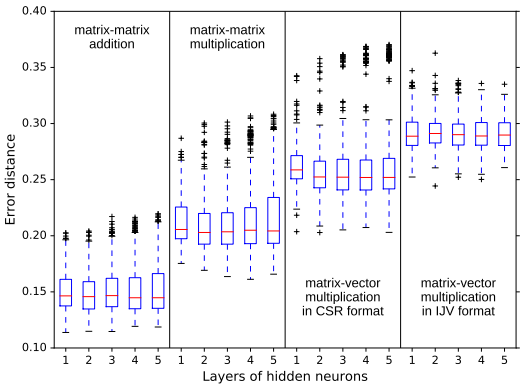


Figure 2: Impact of neuron layers on prediction accuracy of NN.

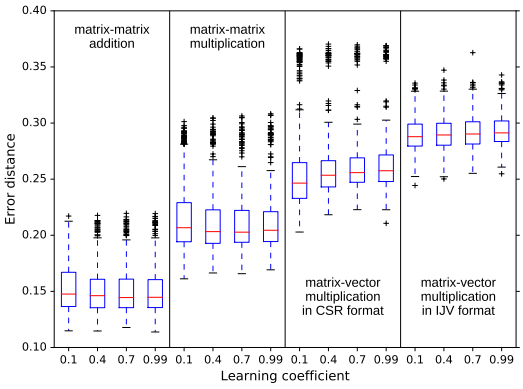


Figure 3: Impact of learning coefficients on prediction accuracy of NN.

where $Err_t = P_t - M_t$, $Err_c = P_c - M_c$, $Err_u = P_u - M_u$, $Err_e = P_e - M_e$, P_t is predicted number of threads, M_t is measured number of threads, P_c is predicted core frequency, M_c is measured core frequency, P_u is predicted uncore frequency, M_u is measured uncore frequency, P_e is predicted relative energy savings and M_e is measured relative energy savings. P values were predicted by a neural network on the output neurons, M values were experimentally measured on the cluster. All predicted and measured values were normalised.

For the following Figs. 1, 2, 3 and 4, only 1 000 epoch runs were plotted to ensure fully trained networks except for the SpMVmultIJV, where more than 100 training cycles led to overtraining. More than 1 000 epochs (not shown in the plots) did not improve the error distance. Every box of each boxplot represents all variants of neurons, layers, learning coefs. and training set sizes. One of these attributes is always set on the X axis of each chart.

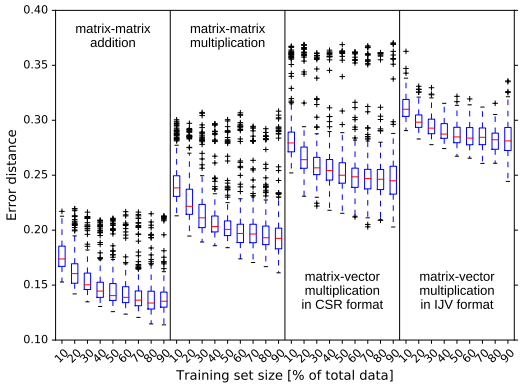


Figure 4: Impact of training on prediction accuracy of NN.

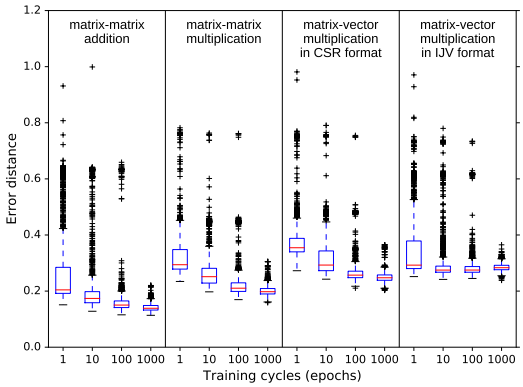


Figure 5: Impact of the training cycles on prediction accuracy of NN.

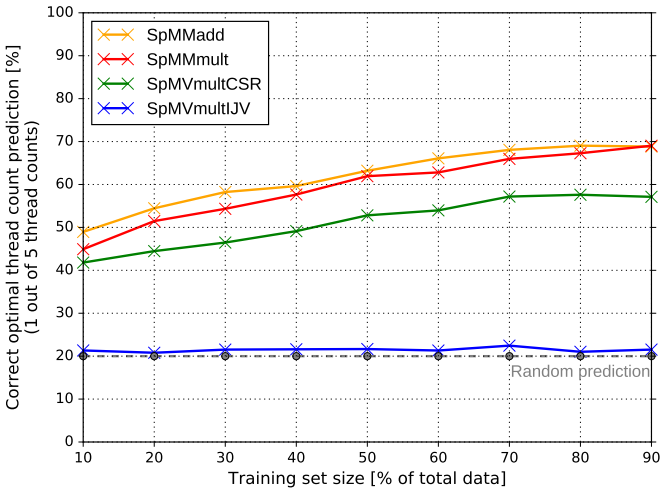
Fig. 1 shows that about 30 hidden neurons is enough to represent the relation between the input and output data. The number of hidden layers (see Fig. 2) was not detrimental to the results and 1–2 layers produced best mean results. The learning coefficient (see Fig. 3) did not have a big impact either, the best value depends on the algorithm. The number of training cycles (see Fig. 5) was an important factor in ensuring the network has converged to an optimal state.

The most important attribute turned up to be the training set size. It is much less time and resource-consuming to train a bigger network rather than enlarging the training set, which always improved the prediction quite significantly (see Fig. 4).

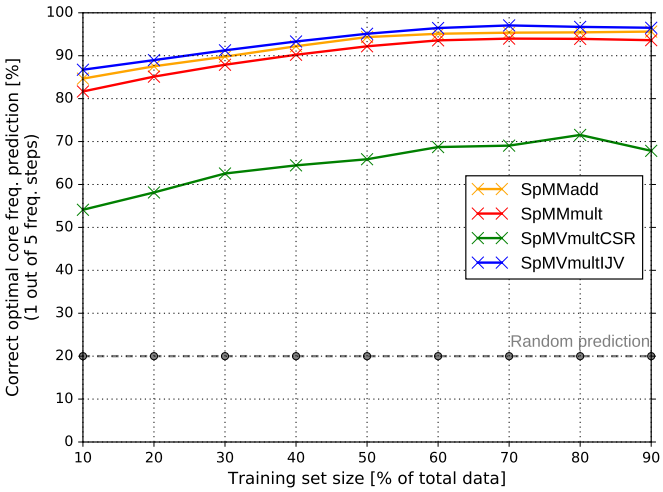
Since the network training takes a negligible amount of time compared to the sparse calculations on average matrices,

 ISBN: 978-1-61208-655-2

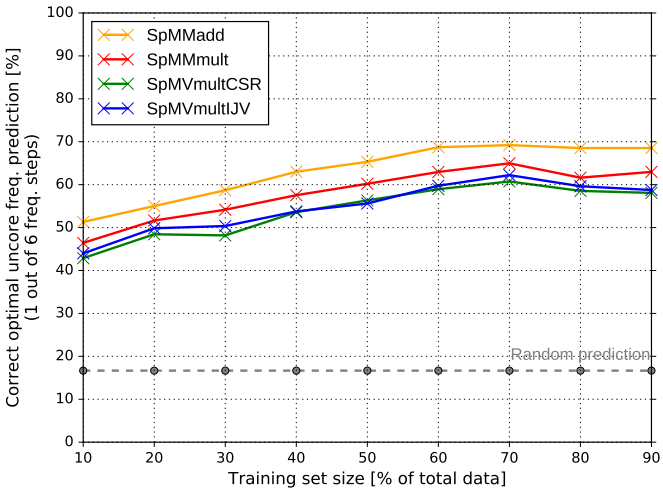
(a) Number of threads



(b) Core frequency



(c) Uncore frequency



(d) Total energy

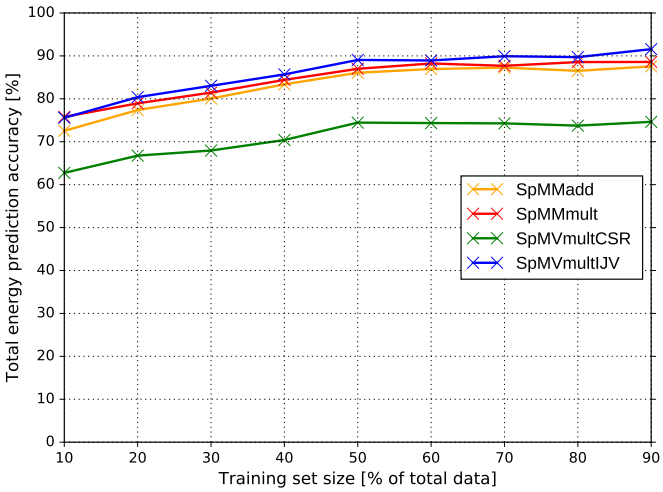


Figure 6: Prediction accuracy of the values back to absolute discrete values.

the best network was chosen for each sparse algorithm (see Table I) and the effect of a training set size on prediction accuracy is more closely analysed in Fig. 6. The Y axis represents the percentage of correctly predicted values after rounding each one to the closest discrete value experimentally measured on the cluster. For example, if 6 is the optimal number of threads and the neural network predicted any value within the $[5, 7)$ range, the prediction was considered correct. The same applies to the core and uncore frequencies. The total energy plot represents the average accuracy of predicted Joules compared to measured Joules.

Note that increasing the number of layers and neurons over 30 has negligible impact on optimal values, so if the neural network learning effort was also considered important, a neural network with 1 layer of 30 hidden neurons provided similarly good results in this case.

The thread count prediction accuracy ranges from 40 to 70% except for the SpMVmultIJV algorithm, which has a success rate of a random prediction. The main reason is that no thread count provided unambiguously best energy savings, more than one setting was often very close to the optimal one, so the neural network had more trouble learning the best value. The same behaviour was observed with the uncore frequency. The distribution for threads and uncore freq. was scattered across the whole spectrum of values, on the other hand, the optimal

TABLE I. BEST NEURAL NETWORK SETTINGS.

Algorithm	Neurons	Layers	Learning coef.	Epochs
SpMMadd	100	1	0.99	1 000
SpMMmult	250	4	0.1	1 000
SpMVmultCSR	50	5	0.1	1 000
SpMVmultIJV	50	2	0.1	100

 ISBN: 978-1-61208-655-2

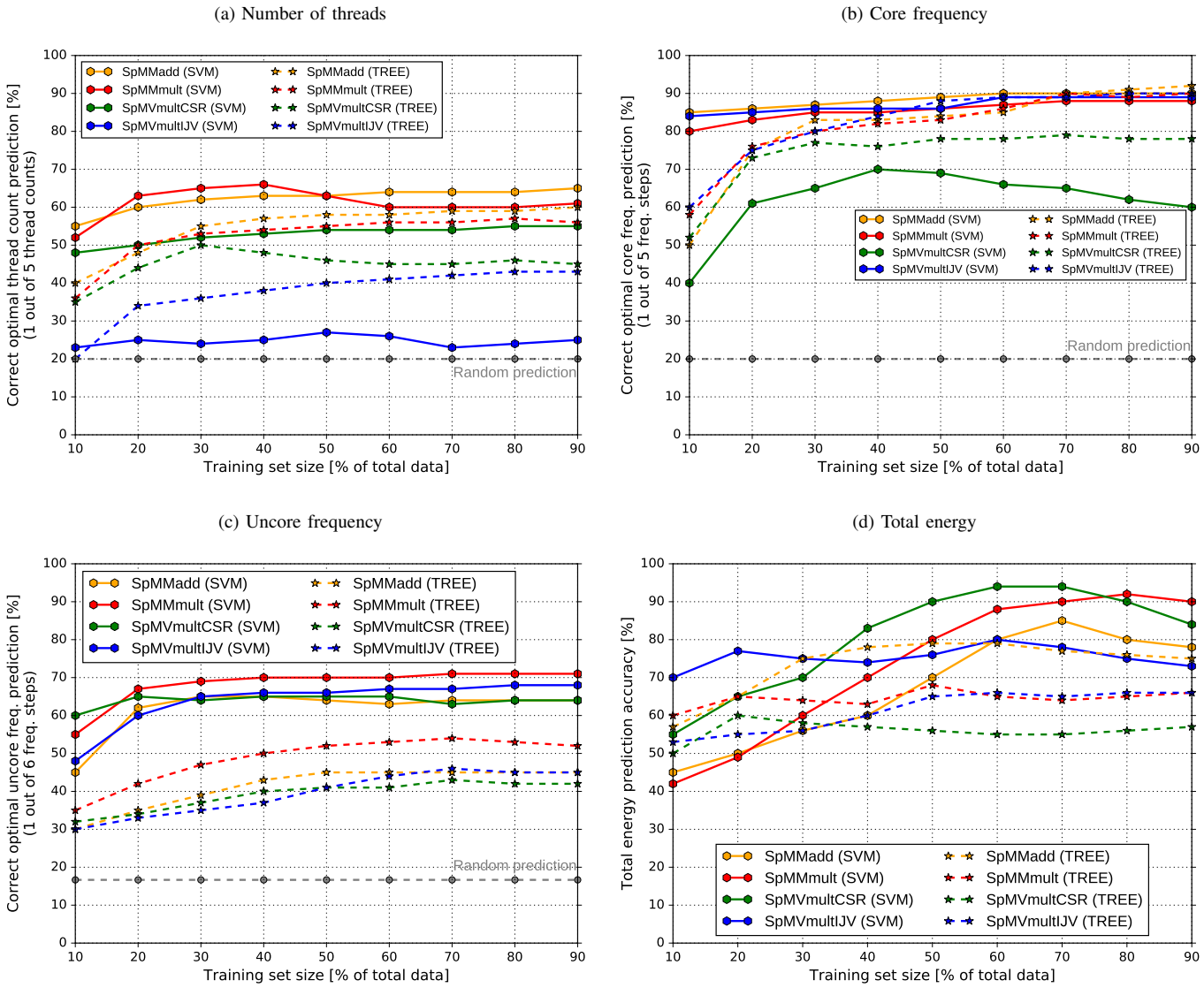


Figure 7: Prediction accuracy of Support Vector Machine (SVM in the legends) and Fast decision tree learners (TREE in the legends).

core frequency was 2.5 GHz in about 90% of the runs and quite rarely 2.1 GHz or even lower, so the prediction was quite simple, except for the SPMVmultCSR, where the values are more distributed into the lower frequencies around 1.5 GHz. Energy was also predicted quite precisely with up to 90% accuracy.

Fig. 7 similarly shows the prediction accuracy using support vector machines (SVM) and fast decision tree learners (TREE). SVM provide results similar to neural networks, on average the precision is 5–10% worse. The worst results are achieved in the same spots, predicting the number of threads of the SPMVmultIJV algorithm and the core frequency of the SPMVmultCSR, which only strenghten the argument that the current data configuration is hard to predict. However, TREE provides noticeably better results in these area and was able to find some additional dependencies. On average, SVM is about 5–10% more accurate than TREE and neural networks are 5–10% more accurate than SVM.

IV. Conclusion

The ability of neural networks, support vector machines and fast decision tree learners to predict the energy requirements and optimal hardware parameters (number of threads and core and uncore frequencies) for the best energy-to-solution of sparse matrix operations was evaluated. The experiments showed that 1 hidden layer of 30 neurons, 1000 learning cycles and almost any learning coefficient were able to produce results close the best achievable. The prediction accuracy depends on the algorithm and the predicted parameter, and for neural networks can reach up to 95%, with 60-70% being the average for most combinations. Support vector machines were 5–10% less accurate in their prediction compared to neural networks, and struggled on similar problems. Fast decision

 ISBN: 978-1-61208-655-2

tree learners were 5–10% less accurate than SVMs, however, they proved to be useful in specific areas. The tree structure might be more appropriate in certain situations. Overall, all algorithms proved to be a useful tool in the area of energy efficiency.

ACKNOWLEDGMENT

This work was supported by the READEX project - the European Union's Horizon 2020 research and innovation programme under grant agreement No. 671657.

This work was supported by The Ministry of Education, Youth and Sports from the Large Infrastructures for Research, Experimental Development and Innovations project 'IT4Innovations National Supercomputing Center - LM2015070'.

This work was supported by the FIT-S-17-3994 Advanced parallel and embedded computer systems project.

REFERENCES

[1] A. von Gladi, M. Ahlborg, T. Knopp, and T. M. Buzug, "Compressed sensing of the system matrix and sparse reconstruction of the particle concentration in magnetic particle imaging," *IEEE Transactions on Magnetics*, vol. 51, no. 2, pp. 1–4, Feb 2015.

[2] U. Becciani, E. Sciacca, M. Bandieramonte, A. Vecchiato, B. Bucciarelli, and M. G. Lattanzi, "Solving a very large-scale sparse linear system with a parallel algorithm in the gaia mission," in *2014 International Conference on High Performance Computing Simulation (HPCS)*, July 2014, pp. 104–111.

[3] J. S. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization," in *Advances in Neural Information Processing Systems 24*, J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2011, pp. 2546–2554. [Online]. Available: http://papers.nips.cc/paper/4443-algorithms-for-hyper-parameter-optimization.pdf

[4] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *J. Mach. Learn. Res.*, vol. 13, no. 1, pp. 281–305, Feb. 2012. [Online]. Available: http://dl.acm.org/citation.cfm?id=2503308.2188395

[5] D. Stamoulis, E. Cai, D. Juan, and D. Marculescu, "Hyperpower: Power- and memory-constrained hyper-parameter optimization for neural networks," *CoRR*, vol. abs/1712.02446, 2017. [Online]. Available: http://arxiv.org/abs/1712.02446

[6] NVIDIA. Tegra X1 processor. URL: http://www.nvidia.com/object/tegra-x1-processor.html [accessed: 2018-07-10].

[7] S. C. Smithson, G. Yang, W. J. Gross, and B. H. Meyer, "Neural networks designing neural networks: Multi-objective hyper-parameter optimization," in *2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, Nov 2016, pp. 1–8.

[8] L. Kotthoff, C. Thornton, H. Hoos, F. Hutter, and K. Leyton-Brown, "Auto-weka 2.0: Automatic model selection and hyperparameter optimization in weka," vol. 18, pp. 1–5, 03 2017.

[9] M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, and F. Hutter, "Efficient and robust automated machine learning," in *Advances in Neural Information Processing Systems 28*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, Eds. Curran Associates, Inc., 2015, pp. 2962–2970. [Online]. Available: http://papers.nips.cc/paper/5872-efficient-and-robust-automated-machine-learning.pdf

[10] T. A. University. SuiteSparse Matrix Collection. URL: https://sparse.tamu.edu/ [accessed: 2018-07-10].

[11] T. A. Davis and Y. Hu, "The University of Florida Sparse Matrix Collection," *ACM Trans. Math. Softw.*, vol. 38, no. 1, pp. 1:1–1:25, Dec. 2011. [Online]. Available: http://doi.acm.org/10.1145/2049662.2049663

[12] Genann. Genann repository. URL: https://github.com/codeplea/genann [accessed: 2018-07-10].

[13] A. Gulli and S. Pal, *Deep Learning with Keras*. Packt Publishing, 2017. [Online]. Available: https://books.google.cz/books?id=20EwDwAAQBAJ

[14] Technische Universitat Dresden. System Taurus. URL: https://doc.zih.tu-dresden.de/hpc-wiki/bin/view/Compendium/SystemTaurus [accessed: 2018-06-25].

[15] O. Vysocky, M. Beseda, L. Riha, J. Zapletal, V. Nikl, M. Lysaght, and V. Kannan, "Evaluation of the HPC applications dynamic behavior in terms of energy consumption," in *Proceedings of the Fifth International Conference on Parallel, Distributed, Grid and Cloud Computing for Engineering*, paper 3, 2017. doi:10.4203/ccp.111.3.

[16] P. Ivanyi, B. H. V. Topping, and G. Varady, Eds., *Proceedings of the Fifth International Conference on Parallel, Distributed, Grid and Cloud Computing for Engineering*. Civil-Comp Press, Stirlingshire, UK.

[17] D. Hackenberg, T. Ilsche, J. Schuchart, R. Schne, W. E. Nagel, M. Simon, and Y. Georgiou, "HDEEM: High definition energy efficiency monitoring," in *2014 Energy Efficient Supercomputing Workshop*, Nov 2014, pp. 1–10.

 ISBN: 978-1-61208-655-2