Exploring the Accuracy – Energy Trade-off in Machine Learning

Alexander E.I Brownlee, Jason Adair, Saemundur O. Haraldsson and John Jabbo
Computing Science and Mathematics
University of Stirling, Scotland, UK
Email: sbr@cs.stir.ac.uk

***Abstract*—Machine learning accounts for considerable global electricity demand and resulting environmental impact, as training a large deep-learning model produces 284 000kgs of the greenhouse gas carbon dioxide. In recent years, search-based approaches have begun to explore improving software to consume less energy. Machine learning is a particularly strong candidate for this because it is possible to trade off functionality (accuracy) against energy consumption, whereas with many programs functionality is simply a pass-or-fail constraint. We use a grid search to explore hyperparameter configurations for a multilayer perceptron on five classification data sets, considering trade-offs of classification accuracy against training or inference energy. On one data set, we show that 77% of energy consumption for inference can be saved by reducing accuracy from 94.3% to 93.2%. Energy for training can also be reduced by 30-50% with minimal loss of accuracy. We also find that structural parameters like hidden layer size is a major driver of the energy-accuracy trade-off, but there is some evidence that non-structural hyperparameters influence the trade-off too. We also show that a search-based approach has the potential to identify these trade-offs more efficiently than the grid search.**

I. Introduction

A crucial part of developing a machine learning platform for a given task is configuring and tuning the model to maximise performance. Typically this targets one objective: maximising the accuracy of the final model on unseen data. However, little thought is usually given to the energy consumed by the model, either in training or inference. This matters at both extremes of scale. As datacentres are predicted to consume 8–21% of global electricity by 2025, it is estimated that training a large deep-learning model produces 284 000kgs of the greenhouse gas carbon dioxide, equivalent to the lifetime emissions of five cars [1]. On modern smartphones, battery life is a critical aspect of user experience and while display, wifi and GPS also have a large impact on power consumption, CPU usage is a major factor, consuming 27–35% of the device's power [2].

The machine learning community is becoming aware of this as an issue, with a few groups exploring measurement of energy for deep learning [3]–[6], techniques to reduce model complexity [7]–[9], and the Low Power Image Recognition Challenge (LPIRC) [10]. It would seem that extending the conventional approaches to parameter tuning of machine learning models to include energy efficiency would be a potential easy win towards improvements in this area.

Search-based approaches have proven successful for targeting non-functional properties of software (including energy) in other areas, e.g., [11]–[16]. Machine learning is an ideal target for this approach, because functionality (i.e. model accuracy) is measured on a continuum rather than a fixed specification. So the opportunity exists to trade-off a small part of functionality to gain in another area such as CPU resources consumed. The authors are aware of only three studies [17]–[19] applying search-based techniques to tuning of machine learning for energy using model-based approaches to estimating energy consumption; in Java for multilayer-perceptrons [17], and in Python for deep neural networks [18], [19]. The latter two only focused on structural parameters on the assumption that these directly impact energy; yet these do affect accuracy and here, as we are considering the trade-off between accuracy and energy, we consider both.

The interesting result in [17] was that, while occasionally a trade-off existed between model accuracy and energy consumption, for several datasets it was possible to optimise both. That is, there was no trade-off between the two. In the present paper, as well as using a different environment (Python scikit-learn rather than Java WEKA), we also consider more hyperparameters and it will be interesting to see if the same conclusion can be made.

Motivated by the above, the present study further explores the possibility of hyperparameter tuning for energy-efficient machine learning. We focus on the popular scikit-learn library in Python, using the RAPL tool provided by Intel CPUs to measure energy consumption. We wish to investigate the following questions:

- RQ1 Is there a trade-off between energy efficiency and model accuracy?
- RQ2 What are the crucial factors driving energy use?
- RQ3 Is there a difference in these factors between energy use for training (building the model) and for inference (using the model)?
- RQ4 Are there some models where we can spend more energy training to get a more efficient inference stage?
- RQ5 What is the potential for search-based optimisation in this context?

This preliminary study answers RQ1 in the affirmative: there is a trade-off apparent for the models and data sets explored. On one data set, we show that it is possible to reduce the energy consumption for inference by 77%, with accuracy only dropping from 94.3% to 93.2%. Energy savings of 30-50% are also possible for training, again with minimal reductions in classification accuracy of the resulting model.

We also make some preliminary investigations towards answering RQ2-5. As one might expect, structural parameters like the number of hidden layer neurons are important, but there is also some evidence that non-structural hyperparameters have some influence on the trade-off as well. There are clear differences in the trade-offs when considering training or inference energy. We also show that a search-based approach is able to approximate the trade-off reasonably well with fewer function evaluations than a grid search.

II. Target of Investigation

We focus on one of the most popular paradigms in machine learning: multilayer perceptrons (MLP). Our experiments used the MLPClassifier implementation within scikit-learn 0.23.1, provided by Anaconda 4.9.2 running Python 3.8.3. MLPs were also studied in [17] using the WEKA framework.

The hyperparameters chosen for variation in the study are listed in Table I. This is a comprehensive list of all hyperparameters that are open to tuning in the scikit-learn MLPClassifier API, except those that are specific to only a subset of solver algorithms. Please note that the range of hyperparameters used to tune the MLP on the 'mortgage' dataset was restricted due to the prohibitive computational cost. The possible values are arbitrary but based on typical values used in real applications and always include the default chosen by scikit-learn's developers. A brief note is given of the impact of each hyperparameter: further detail can be found in the scikit-learn documentation[1].

We are interested in the energy consumption attributable to both the training and inference stages of the model's life cycle. Some of the aforementioned hyperparameters clearly have an impact on the final structure of the model and the processing involved in inference (e.g., hidden layer sizes), and so are likely to have a direct impact on energy during inference. While some (e.g., max iterations for the training algorithm) only directly impact the training process, they will typically interact with other hyperparameters that are relevant to the inference process [20]: for example, it might be that reducing the hidden layer size reduces energy, but then requires a different activation function to improve the accuracy. As such, we include all hyperparameters in the explorations for both stages.

III. Methodology

We now describe the tools used within this study. Our experiments make use of PyRAPL [2], which provides a Python interface to the Intel "Running Average Power Limit" (RAPL) technology. RAPL estimates power consumption of a CPU and is available on Intel CPUs since the Sandy Bridge generation, albeit with different levels of energy information available. On the Xeon CPUs used in this study, RAPL provides measurements for the energy consumption of the whole CPU socket package. Although RAPL is actually a model-based estimator built in to the CPU, its estimates have been shown to be highly correlated with measured power consumption [21].

The experiments were performed using a workstation running Debian OS, with two 16-core Intel Xeon E5-2620 v4 processors, each running at 2.1 GHz (boosting to 3GHz), with 32GB DDR4 memory running at 2.4GHz. The experiments were run sequentially, and no other computationally intensive processes were run in parallel with the experiments.

Our experiments apply the models to five data sets, listed in Table II. As with [17], we used four of the well-known UCI classification benchmarks (diabetes, glass, ionosphere, iris). As these data sets are rather small compared to many practical applications, we also included a mortgage lending data set from Kaggle[3]. This contains records and application status of 500 000 mortgage loan applications from 6111 different lending institutions in the USA. In common with most real-world data sets, the mortgage data needed some pre-processing before modelling. Specifically, five variables with flat and wide distributions (essentially just random noise from the modelling perspective) were dropped. These were row_id, lender, msa_md, state_code, and county_code. Instances with outlier values were removed where loan_amount$\leq$ 3750, applicant_income $<=$ 2000, and population $<=$ 35000. Categorical variables were treated with one-hot encoding, and numerical variables were scaled to the range [0,1].

For all the data sets, the original data was split into training and test sets with a ratio of 75:25.

IV. Experiment 1: Grid Search

Our first experiment applied a grid search (exhaustive exploration) over the possible hyperparameters values for each model type on each data set. Grid search is commonly used for exploring hyperparameter configurations, and although it is limited to a relatively low-resolution search of the space through having a finite set of values for each hyperparameter, it has the advantage of being immune to any randomness in the search process. The results from this search were post-processed to find the Pareto-optimal (trade-off) configurations in terms of different metrics. As the search is exhaustive these configurations are genuinely Pareto-optimal for the range of hyperparameter values tried. This experiment is designed to answer the first research question, at least in the context of MLPs applied to these specific data sets: *Is there a trade-off between energy efficiency and model accuracy?*

In the grid search experiment, we measured the following for each configuration:

- Test Accuracy: the accuracy of the model on the unseen test data set
- CV Accuracy: the mean accuracy over 5 folds of cross-fold validation on the training data

[1]https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html

[2]Version 0.2.3.1 https://pypi.org/project/pyRAPL/

[3]https://www.kaggle.com/dinu1763/mortgage-loan-approval

TABLE I
THE MLP HYPERPARAMETERS AND RANGE OF VALUES USED IN OUR EXPERIMENTS

Hyperparameter	Range / values (UCI data)	Range / values (Mortgage data)	Description
Hidden layer size	2, 5, 10, 20, 50, 100, 200, 500	50, 100, 200	Number of neurons in hidden layer
Activation fn	logistic, tanh, relu	logistic, tanh, relu	Activation function for the hidden layer neurons
Solver	sgd, adam, lbfgs	sgd, adam	Solver algorithm used to fit network weights
Alpha	0.002, 0.001, 0.0005, 0.0002, 0.0001	0.0005, 0.0001	L2 penalty (regularization term) parameter
Max iterations	100, 200, 400, 800	200, 400	The solver iterates until convergence or this number of iterations

Dataset	Instances	Attributes
Pima Indians Diabetes	768	8
Glass Identification	214	10
Ionosphere	351	34
Iris	150	4
Mortgage	500 000	17

TABLE II
DATASETS USED

- Training CPU Energy: the energy use in microjoules for the CPU reported by PyRAPL for model fitting
- Training CPU Duration: the time taken to fit the model to the training data, measured in seconds with nanosecond resolution using python's `time.process_time()` method
- Training Runtime: the wall-clock time taken to fit the model to the training data, measured using python's `time.time()` method
- Inference CPU Energy: the energy use for the CPU reported by PyRAPL for 1000 applications of the model to the test data
- Inference CPU Duration: the time taken to apply the model to the test data 1000 times, measured using python's `time.process_time()` method
- Inference Runtime: the wall-clock time taken to apply the model to the test data 1000 times, measured using python's `time.time()` method

The measurements for inference (application of the models to the test data) were repeated 1000 times, as inference is much less CPU intensive than training, resulting in much smaller values to measure for each metric.

In reporting our experiments, for brevity here we focus our reporting on energy measurements and the model accuracy for the unseen test data set. The full set of results and plots, including CPU times and cross-fold validation accuracy (which used the training data) can be found at the URL given at the end of the paper.

To briefly summarise the results not detailed in the paper: similar trade-offs do appear for CPU time and run time vs the accuracy measures. Similarly, changing to cross-fold validation accuracy rather than test accuracy shows similar trade-offs, although the shapes of each Pareto front are subtly different.

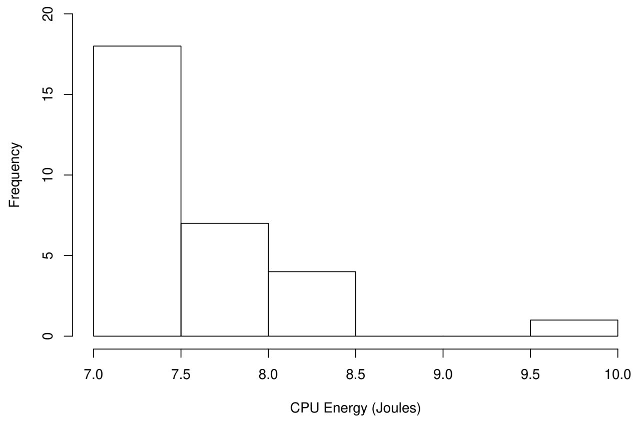


Fig. 1. Histogram of CPU energy for model training of one configuration on Iris, over the 30 repeat runs. Most look like this: clearly not normal, so we use non-parametric statistics in our reporting.

A. CPU time and energy

Run times, and to an even greater extent, energy measurements [22] are subject to noise from the numerous environmental factors affecting a running system. This includes varying CPU speeds, caching, and other processes running. As such, for all of the experiments, we repeated the training and test runs 30 times. These repeats were also interleaved so that, as much as possible, environmental effects would impact the results for all configurations evenly, as summarised in Algorithm 1. These measurements across repeat runs do not usually follow a normal distribution: Figure 1 summarises the 30 energy measurements for training the MLP model on the Iris data set. As such, where the values are aggregated, we use non-parametric statistics such as median and interquartile range. In some rare cases (fewer than 0.05% of runs), PyRAPL failed, returning a negative value for energy. These results were filtered out before our analysis.

Often, CPU time is used as a proxy for energy use [23], but it has been found that this can be inaccurate, particularly because this omits CPU idle states [24]. It would appear that in the present setting, while broadly linear for longer runs, particularly with shorter runs the relationship is not so simple. Figures 2 and 3 display the CPU energy and CPU

```
Data: $M$ sklearn regression model (MLP or RF)
Data: $C$ all hyperparameter configurations of $M$
Data: $Tr$ training data set
Data: $Te$ test data set
1 for $i \leftarrow 1$ to 30 do
2     foreach $c \in C$ do
3         $m \leftarrow M$.copy(); $m \leftarrow m$.setParams($c$);
          accuracy $\leftarrow m$.crossFoldValidation($Tr$);
          trainingMetrics $\leftarrow$
          measureEnergyAndTime($m$.fit($Tr$));
4         for $j \leftarrow 1$ to 1000 do
5             testMetrics $\leftarrow$
                measureEnergyAndTime($m$.predict($Te$));
6         end
7     end
8 end
```

Algorithm 1: Interleaving of runs

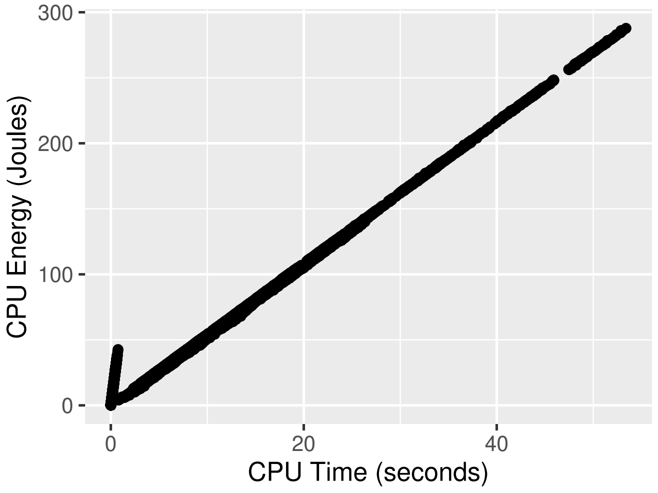


Fig. 2. CPU time vs CPU energy (training, Iris dataset).

time measurements from all hyperparameter configurations for the training and inference stages of the MLP on the Iris dataset. The shorter running training and inference stages both show a separate linear relationship with energy to the longer runs; possibly due to the effects of caching and Intel Turbo Boost. This motivates taking separate measurements specific to energy. In this paper we will report energy measurements only from here on, although the publicly available results do include the CPU and run times as well.

B. Grid results

We plotted each of the time and energy metrics against the accuracy on the unseen test data set (ultimately what we are most interested in from a modelling perspective – as noted above, we also measured cross-fold validation accuracy and it would appear that similar trends exist for that). Time and energy measurements across the 30 repeat runs for each configuration were aggregated, and the points on each plot show the median value for each metric, with error bars indicating the interquartile range. The ideal configuration would be located bottom-right, with minimal run-time or energy consumption, but maximal accuracy. We might expect a trade-off between these metrics to span bottom-left (low energy / run-time and low accuracy) to top-right (high energy / run-time and high accuracy). One of these plots is given in Figure 4.

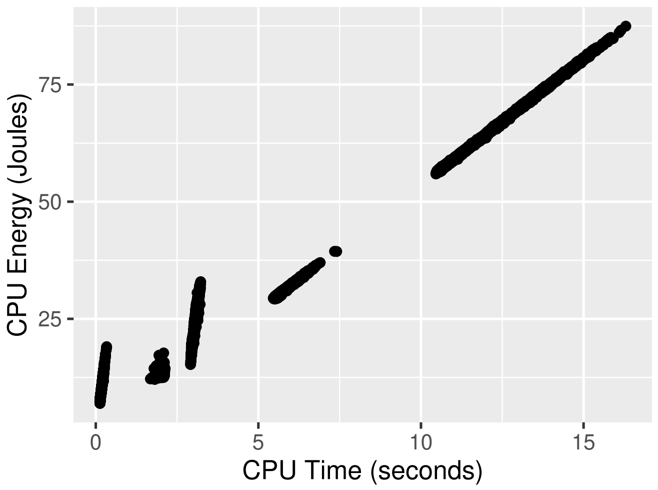


Fig. 3. CPU time vs CPU energy (inference, iris dataset).

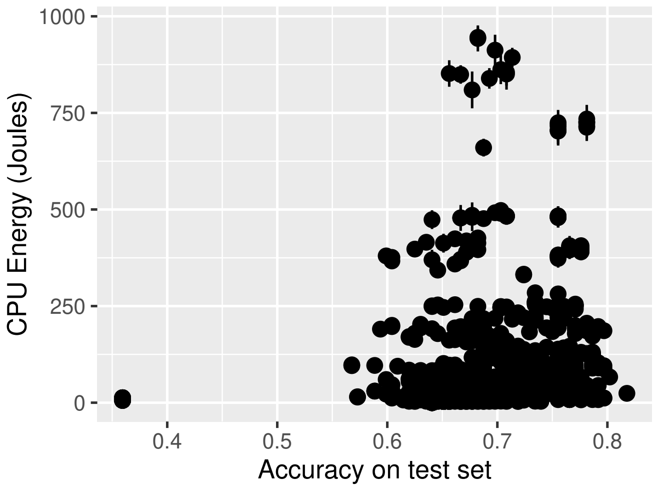


Fig. 4. For all configurations of the MLP on the Diabetes data set, accuracy on test set vs CPU energy while training.

The full set of results can be seen via the URL provided at the end of the paper. We will summarise the overall trends seen here. For the energy measurements on training and inference phases, on all data sets, there is a broad spread of configurations in both accuracy and energy. Overall, it would seem that, against the full range of possible accuracy values and energy measurements, that it is possible to reach relatively high accuracy with relatively low energy consumption. However, a trade-off does seem to appear in the high-energy high-accuracy region, so the next section will investigate this more closely.

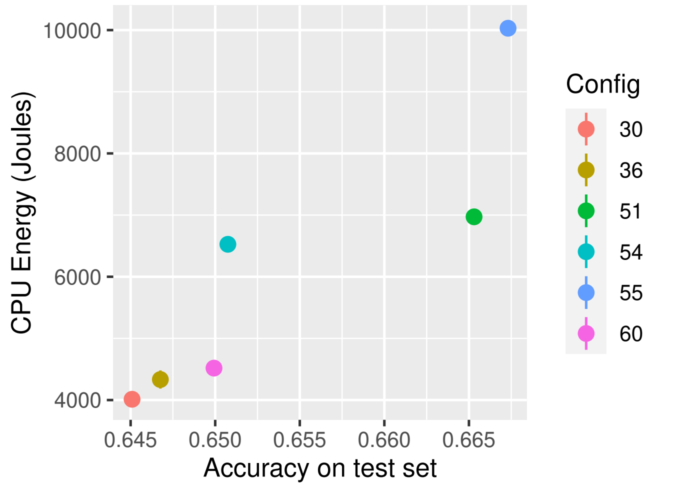


Fig. 5. Pareto Front: MLP on the Mortgage data set. The trade-off of model accuracy on test data vs CPU energy for training. Note, the errors bars are still present here, but small enough to be obscured by the points themselves.

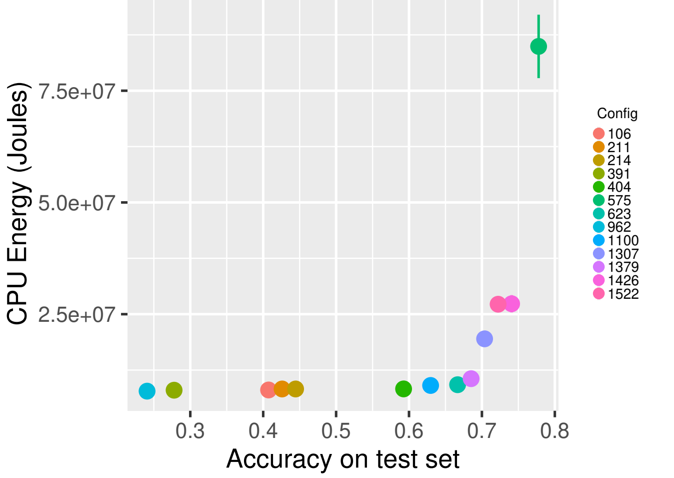
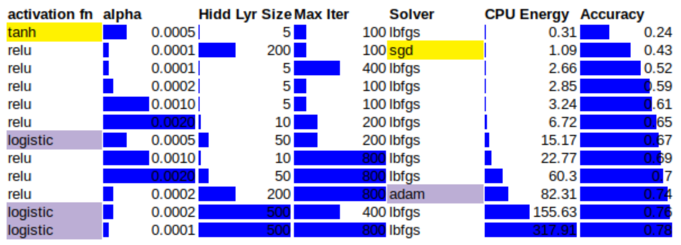


Fig. 6. Pareto Front: MLP on the Glass data set. The trade-off of model accuracy on test data vs CPU energy for inference.



activation fn	alpha	Hidd Lyr Size	Max Iter	Solver	CPU Energy	Accuracy
tanh	0.0005	5	100	lbfgs	0.31	0.24
relu	0.0001	200	100	sgd	1.09	0.43
relu	0.0001	5	400	lbfgs	2.66	0.52
relu	0.0002	5	100	lbfgs	2.85	0.59
relu	0.0010	5	100	lbfgs	3.24	0.61
relu	0.0020	10	200	lbfgs	6.72	0.65
logistic	0.0005	50	200	lbfgs	15.17	0.67
relu	0.0010	10	800	lbfgs	22.77	0.69
relu	0.0020	50	800	lbfgs	60.3	0.7
relu	0.0002	200	800	adam	82.31	0.74
logistic	0.0002	500	400	lbfgs	155.63	0.76
logistic	0.0001	500	800	lbfgs	317.91	0.78

Fig. 7. Hyperparameter trends along the Pareto front representing the trade-off of test accuracy vs training energy for the MLP on the Glass data set

C. Results: trade-off

To inspect the utopian region (high accuracy, low energy) more thoroughly, we filtered the configurations to those in the Pareto-optimal set. To find these, we took those configurations explored by the grid search, and removed any for which there existed a configurations with both lower accuracy and higher energy. This process was repeated for both CPU energy during training and CPU energy during inference.

Figure 5 shows the CPU energy consumed for training the MLP on the Mortgage data set, demonstrating a clear trade-off. Importantly, at the high-accuracy end, there is jump in energy consumption with only a small increase in accuracy (known as a ”knee point”). This was observed for the MLP on four of the five data sets studied (the exception being Iris). On the Mortgage data set, a slight reduction in accuracy (66.7 to 66.5%) reduces energy consumption by 30.4%. With the Glass data set, dropping accuracy from 77.8% to 75.9% reduces energy consumption by 51%.

Figure 6 shows that the same can be observed for the inference stage with the MLP on the Glass data set. A trade-off was still observed for each of the data sets, though the knee point was really only apparent with Glass and Ionosphere for the inference stage. However, the jump is very large for Ionosphere: reducing accuracy from 94.3% to 93.2% reduces energy consumption for inference by 77%.

V. Experiment 2: Parameter relationships

The second experiment is designed to explore the relationship between the hyperparameters and the energy / accuracy metrics. This makes a step towards answering Research Question 2: *What are the crucial factors driving energy use?*.

In this experiment, we took the Pareto fronts from the previous section, and sorted the configurations in ascending order of energy and accuracy. These were then plotted as illustrated in Figure 7. Here, the first five columns are the hyperparameter values (with either a unique colour for each value with the categorical hyperparameters such as ”solver”, or a bar showing the value for the numerical hyperparameters). The final two columns show the CPU energy and the accuracy, to help identify where particular hyperparameters values correspond with knee-points in energy use.

After exploring these plots for accuracy vs energy use in both training and inference, we can make some general observations for the data sets and hyperparameters we explored. For test accuracy vs energy for training:

- ‘tanh’ is often at the high energy+accuracy end, otherwise no clear pattern shows for activation functions or alpha
- Hidden layer size clearly increases with energy+accuracy for the Glass data, but otherwise there is no strong pattern
- Counter-intuitively, sometimes we get higher accuracy with a smaller hidden layer size and longer training. This would partly explain why we sometimes get more energy on training and less for inference (especially true with Diabetes and Iris).
- ‘adam’ solver typically appears in high energy high accuracy configurations; ‘sgd’ typically appears in low energy low accuracy configurations

For test accuracy vs energy for inference:

- There is no clear pattern for activation function or alpha.

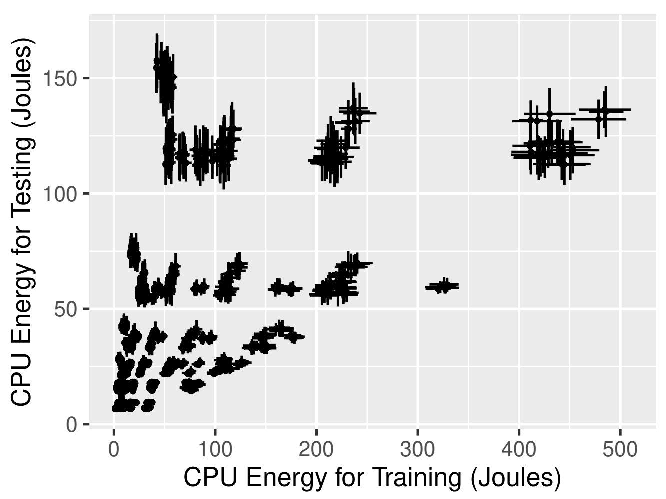


Fig. 8. CPU energy (Joules) consumed for each configuration, in training and inference, on the Ionosphere data set. For each point, the horizontal bar is the interquartile range in training energy, and the vertical bar is the IQR in inference (testing) energy. The crossover point is the median for both values.

- Unsurprisingly, here hidden layer size is important. There is a clear positive correlation with energy and accuracy.
- There is a much less clear pattern for maximum iterations.
- There is also a less clear pattern for solver, though 'adam' is often at the low energy+accuracy end.
- The leap in energy consumption on the Pareto front for ionosphere is going from 2 to 100 hidden neurons.

These results also go some way towards answering RQ3: *Is there a difference in these factors between energy use for model training and for inference?* Clearly, the answer is yes. It is also particularly interesting to see some evidence that the choice of solver has an impact on the trade-off, even for inference, given that it only affects the training process.

We also consider RQ4: *Are there some models where we can spend more energy training to get a more efficient inference stage?* This would be particularly beneficial for deployment where inference is expected to make up the bulk of computational effort. Figure 8 shows the CPU energy measured for inference plotted against the CPU energy measured for training for all configurations of the MLP on the Ionosphere data set. All the other data sets showed the same general picture. At least for these results, it would appear that, while it is possible to have model configurations with high energy consumption for inference but low energy for training, there is little evidence that configurations exist where the converse is true. Configurations that consume little energy for inference also appear to be cheap to train.

VI. Experiment 3: Application of NSGA-II to Approximate the Trade-offs

All the above experiments and results used an exhaustive grid search. It is well-known that stochastic searches offer scope to efficiently explore a much wider range of hyperparameter values with a similar cost. This leads us to Research Question 5: *What is the potential for search-based optimisation in this context?* We used the same parameters and ranges as for the grid search, but allowed the numerical parameters to take any value between the lower and upper bounds in Table I. The values of the categorical hyperparameters were encoded as integers so the same crossover and mutation operators could be used for all variables. The objectives targeted for optimisation were accuracy on the test data set, and median CPU energy for 30 repeats of model training. The search was performed by NSGA-II as provided by jMetalPy [25][4]. NSGA-II was configured with a population size of 8, polynomial real mutation (rate 1/n), simulated binary crossover (rate 1), and a maximum of 240 function evaluations (arbitrarily chosen as 1/6 the total number of possible configurations, to achieve a reasonable reduction in run-time over the grid search). These parameters could be tuned further, but the point of this experiment is simply to show the potential for a search-based algorithm on this problem: population size was chosen to allow a reasonable number of generations within the tight evaluation budget; the other parameters were simply the defaults as set within jMetalPy.

Figure 9 shows the minimal, median and maximal attainment curves [26] reached by NSGA-II plotted over the results from the grid search. The median attainment curve (that is, the part of objective space reached by at least half of the repeat runs of the algorithm) is shown as a solid line. Around this line is a shaded area, bounded by the minimal and maximal attainment curves (reached by only one repeat run and by all repeat runs respectively). This is intended to illustrate the spread of results found by the algorithm. It can be seen that in the best case, NSGA-II was able to get very close to the global Pareto front. In the median and worst cases it was not able to find the best configurations either in terms of energy or accuracy, but still came close. Further tuning of NSGA-II (or indeed an alternative MOEA) may be able to improve on this further, though it may not even be necessary unless the range of options presented to the grid search grew much further.

VII. Related work

Research in machine learning has begun to explore the topic of energy consumption, as well as techniques to improve efficiency. Several researchers have modelled or approximated the energy consumption due to deep learning. This typically involves counting the number of operations (e.g., [8]); and can include corrections for different data types involved [27]. Predictive models relating energy to model-specific features like kernel size and number of layers [6], more general hyperparameters [18], [19], or instructions [17] have also been used. Usually model-based approaches have been used for speed, in contrast to the CPU-provided measurements used in the present paper.

Approaches to reducing the energy consumption in deep learning have focused on reducing the complexity of the network in order to reduce the costly calls to retrieve network

[4]v1.5.5

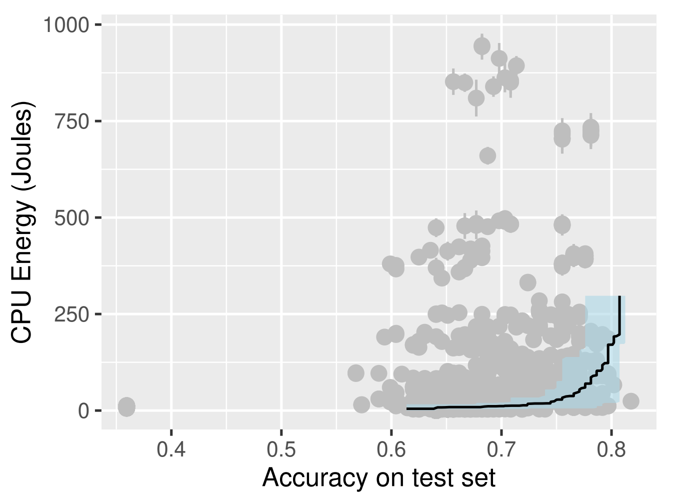


Fig. 9. CPU energy (Joules) for training vs test accuracy, on the Diabetes data set. The points are those found by the grid search. The solid line represents the median attainment curve (the region of objective space reached by half of the runs). The shaded area around each line shows the variation in the runs: the upper extent of shading being the front reached by all runs, and the lower extent of shading being the front reached by only one of the runs.

weights from memory using pruning [8], compression [7] and compact models [9]. Hyperparameter tuning using a GA to identify the energy-accuracy trade-off was briefly explored in [17], and an energy-latency trade-off (with accuracy as a constraint) in [18]. Bayesian optimisation for hyperparameter tuning with energy consumed by a GPU was also considered in [19]. Most studies that have looked at hyperparameters have focused on structural parameters on the assumption that these directly impact energy, with the exception of [17] that did also look at learning rate and momentum. Furthermore, most studies have only focused on reducing energy in the inference stage; only [17]–[19] considered training.

A helpful recent summary of energy in machine learning is found in [3], which reviews techniques for measuring energy and some applications in machine learning. The authors noted that most studies apply post-processing techniques to reduce the energy consumption of the neural network. In general the approach has been to keep the number of model parameters low to reduce the overhead in performance, and very few studies have explicitly incorporated energy-use during training. The authors of [3] also provide two case studies. They looked at four model configurations for data stream mining algorithms, using the Intel power gadget to measure energy, finding that 16% higher accuracy could be obtained at a cost of 1.12x energy. They also looked at the energy consumption of inference in three convolutional neural networks, using a regression model to estimate energy given the number of SIMD instructions and bus accesses called by the model.

More broadly, the energy consumption attributable to software is becoming increasingly important to software developers [28], particularly as software development moves away from the 'mid-range' of the desktop (and the relatively homogeneous architectural configuration of PCs) to the extremes of both large- and small- scale computing [12]. A variety of energy measurement approaches have been described. Often, CPU time is used as a proxy for energy use [23], but this can be inaccurate [24]. Both [29] and [30] use regression models built on the number of a system calls made by a running program. Static or dynamic analysis of program paths with respect to known power consumption of hardware components can also be used [31]. Direct measurement using instrumentation hardware can be in terms of the overall system power [32]–[34]. Power consumption due to the CPU can also be determined via the Intel Power Gadget API [12], [35].

The present paper looks at hyperparameter tuning. The related topic of Deep Parameter Optimization [36], [37] exposes hard-coded constants in source code for tuning; a more general concept is Programming by Optimization [38], where design decisions are moved from software development to nearer deployment, allowing software to be tuned to a particular use-case. Gains can be made in many areas (e.g. functionality, run-time, memory footprint, energy consumption etc.) and the added benefit of improving software is that we might generate multiple alternatives that can easily be hot-swapped in response to changing resource availability or other requirements - termed the Pareto program surface [39]. These alternatives trade-off properties against each other; such as fit-to-target-distribution vs energy use for random number generators [40]. While hardware clearly has a major impact on energy efficiency, applications remain a ripe target for improvement, e.g., 40% energy savings achieved by optimizing workload and thermal management in data centres [41], in addition to the examples above.

VIII. Conclusions

We have explored the trade-off between energy consumption and accuracy for different hyperparameter configurations of a popular machine learning framework, multilayer perceptrons. We have shown that there is a clear trade-off for the models and data sets explored. In one case it was possible to reduce the energy consumption for inference by 77%, with accuracy only dropping from 94.3% to 93.2%. Energy savings of 30-50% for training were possible with minimal reductions in classification accuracy.

We have shown that structural parameters like the number of hidden layer neurons are important drivers of this trade-off when considering both training and inference energy, but there is also some evidence that non-structural hyperparameters have some influence on the trade-off. There are clear differences in the trade-offs when considering training or inference energy, allowing for greater choice in deployment: it can be possible to choose whether to have more energy efficiency in training or in inference. Of course, spending considerable energy on the search for a minimal energy configuration only makes sense if either (a) working on a small but representative sample set; (b) focusing all our efforts on reducing energy for inference. A search-based approach was also shown to be able to approximate the trade-off with fewer function evaluations than a grid search.

The potential for future work in this area is considerable. Clearly, to generalise further we must consider additional data sets and hardware architectures than those explored. There is also very little work done on energy for other popular machine learning paradigms: random forests, linear and logistic regression, k-nearest neighbours etc. Deeper exploration of the hyperparameters that drive the trade-offs is also crucial to allowing us to find more accurate models that do not cost the Earth.

Data Access Statement

The full set of results and visualisations from our experiments can be found at http://hdl.handle.net/11667/173

References

[1] E. Strubell, A. Ganesh, and A. McCallum, "Energy and policy considerations for deep learning in NLP," *CoRR*, vol. abs/1906.02243, 2019.

[2] P. K. D. Pramanik *et al.*, "Power consumption analysis, measurement, management, and issues: A state-of-the-art review of smartphone battery and energy usage," *IEEE Access*, vol. 7, pp. 182 113–182 172, 2019.

[3] E. García-Martín, C. F. Rodrigues, G. Riley, and H. Grahn, "Estimation of energy consumption in machine learning," *Journal of Parallel and Distributed Computing*, vol. 134, pp. 75 – 88, 2019.

[4] C. Rodrigues, G. Riley, and M. Luján, "Synergy: An energy measurement and prediction framework for convolutional neural networks on Jetson TX1," in *Int. Conf. on Parallel and Distributed Processing Techniques and Applications*. CSREA Press, 2018, pp. 375–382.

[5] Y. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE Journal of Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, 2017.

[6] E. Cai, D.-C. Juan, D. Stamoulis, and D. Marculescu, "Neuralpower: Predict and deploy energy-efficient convolutional neural networks," in *Asian Conference on Machine Learning*, 2017, pp. 622–637.

[7] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv:1510.00149*, 2015.

[8] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," *Advances in neural information processing systems*, vol. 28, pp. 1135–1143, 2015.

[9] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and < 0.5 MB model size," *arXiv:1602.07360*, 2016.

[10] K. Gauen, R. Rangan, A. Mohan, Y.-H. Lu, W. Liu, and A. C. Berg, "Low-power image recognition challenge," in *22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2017, pp. 99–104.

[11] D. R. White, "Genetic programming for low-resource systems," Ph.D. dissertation, University of York, UK, 2009.

[12] B. R. Bruce, J. Petke, and M. Harman, "Reducing energy consumption using genetic improvement," in *Proc. GECCO*. Madrid, Spain: ACM, 2015, pp. 1327–1334.

[13] E. Schulte, J. Dorn, S. Harding, S. Forrest, and W. Weimer, "Post-compiler software optimization for reducing energy," in *Proc. Conf. on Architectural Support for Programming Languages & Operating Systems*. New York, NY, USA: ACM, 2014, pp. 639–652.

[14] H. Hoffmann, S. Misailovic, S. Sidiroglou, A. Agarwal, and M. Rinard, "Using code perforation to improve performance, reduce energy consumption, and respond to failures," Massachusetts Institute of Technology, Tech. Rep. MIT-CSAIL-TR-2009-042, 2009.

[15] V. Mrazek, Z. Vasicek, and L. Sekanina, "Evolutionary approximation of software for embedded systems," in *Proc. GECCO Companion*. Madrid, Spain: ACM, 2015, pp. 795–801.

[16] X. T. Nguyen, H. T. Tran, H. Baraki, and K. Geihs, "Optimization of non-functional properties in internet of things applications," *J Netw Comput Appl*, mar 2017.

[17] A. E. I. Brownlee, N. Burles, and J. Swan, "Search-based energy optimization of some ubiquitous algorithms," *IEEE T on Emerg Topics in Comput Intell*, vol. 1, no. 3, pp. 188–201, June 2017.

[18] X. Dai *et al.*, "Chamnet: Towards efficient network design through platform-aware model adaptation," 2018.

[19] D. Stamoulis, E. Cai, D. Juan, and D. Marculescu, "Hyperpower: Power- and memory-constrained hyper-parameter optimization for neural networks," in *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2018, pp. 19–24.

[20] N. Schilling, M. Wistuba, L. Drumond, and L. Schmidt-Thieme, "Hyperparameter optimization with factorized multilayer perceptrons," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 2015, pp. 87–103.

[21] K. N. Khan, M. Hirki, T. Niemi, J. K. Nurminen, and Z. Ou, "Rapl in action: Experiences in using rapl for power measurements," *ACM Trans. Model. Perform. Eval. Comput. Syst.*, vol. 3, no. 2, Mar. 2018.

[22] M. A. Bokhari, M. Wagner, and B. Alexander, "The quest for non-functional property optimisation in heterogeneous and fragmented ecosystems: A distributed approach," in *Proc. GECCO Companion*. Association for Computing Machinery, 2019, p. 1705–1706.

[23] A. Noureddine, A. Bourdon, R. Rouvoy, and L. Seinturier, "Runtime monitoring of software energy hotspots," in *Proc. IEEE/ACM Int'l. Conf. on Automated Softw. Eng.*, Essen, Germany, 2012, pp. 160–169.

[24] C. Zhang, A. Hindle, and D. M. German, "The impact of user choice on energy consumption," *IEEE Softw.*, vol. 31, no. 3, pp. 69–75, 2014.

[25] A. Benítez-Hidalgo, A. J. Nebro, J. García-Nieto, I. Oregi, and J. Del Ser, "jMetalPy: A Python framework for multi-objective optimization with metaheuristics," *Swarm and Evol Comput*, vol. 51, p. 100598, 2019.

[26] J. Knowles, "A summary-attainment-surface plotting method for visualizing the performance of stochastic multiobjective optimizers," in *Proc. Conf. on Intel. Syst. Design and Appl.*, ser. ISDA '05. Washington, DC, USA: IEEE, 2005, pp. 552–557.

[27] T. Yang, Y. Chen, and V. Sze, "Designing energy-efficient convolutional neural networks using energy-aware pruning," in *IEEE Conf on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 6071–6079.

[28] P. Marks, "Let's cut them some slack," *New Scientist*, vol. 230, no. 3067, pp. 36–39, 2016.

[29] K. Aggarwal, A. Hindle, and E. Stroulia, "GreenAdvisor: A tool for analyzing the impact of software evolution on energy consumption," in *Proc. (ICSME)*, Bremen, Germany, 2015, pp. 311–320.

[30] S. A. Chowdhury, L. N. Kumar, M. T. Imam, M. S. M. Jabbar, V. Sapra, K. Aggarwal, A. Hindle, and R. Greiner, "A system-call based model of software energy consumption without hardware instrumentation," in *Int'l Green & Sustainable Comp. Conf.*, Las Vegas, NV, 2015, pp. 1–6.

[31] A. Banerjee, L. K. Chong, C. Ballabriga, and A. Roychoudhury, "Energypatch: Repairing resource leaks to improve energy-efficiency of android apps," *IEEE Trans. Softw. Eng.*, vol. PP, no. 99, pp. 1–1, 2017.

[32] M. Rashid, L. Ardito, and M. Torchiano, "Energy consumption analysis of algorithms implementations," in *ACM/IEEE Int'l Symp. on Empirical Softw. Eng. & Measurement*, Beijing, China, 2015, pp. 82–85.

[33] A. Banerjee, L. K. Chong, S. Chattopadhyay, and A. Roychoudhury, "Detecting energy bugs and hotspots in mobile apps," in *Proc. FSE*. Hong Kong, China: ACM, 2014, pp. 588–598.

[34] A. Hindle, A. Wilson, K. Rasmussen, E. J. Barlow, J. C. Campbell, and S. Romansky, "GreenMiner: a hardware based mining software repositories software energy consumption framework," in *Proc. Mining Software Repositories*. Hyderabad, India: ACM, 2014, pp. 12–21.

[35] H. Field, G. Anderson, and K. Eder, "EACOF: : A framework for providing energy transparency to enable energy-aware software development," in *Proc. ACM SAC*, Gyeongju, Korea, 2014, pp. 1194–1199.

[36] F. Wu, W. Weimer, M. Harman, Y. Jia, and J. Krinke, "Deep parameter optimisation," in *Proc. GECCO*, S. Silva and A. I. Esparcia-Alcázar, Eds. ACM, 2015, pp. 1375–1382.

[37] M. A. Bokhari, B. R. Bruce, B. Alexander, and M. Wagner, "Deep parameter optimisation on android smartphones for energy minimisation: a tale of woe and a proof-of-concept," in *Proc. GECCO Companion*, 2017, pp. 1501–1508.

[38] H. H. Hoos, "Programming by optimization," *Commun. ACM*, vol. 55, no. 2, pp. 70–80, Feb 2012.

[39] M. Harman, W. B. Langdon, Y. Jia, D. R. White, A. Arcuri, and J. A. Clark, "The GISMOE challenge: Constructing the Pareto program surface using genetic programming to find better programs," in *Conf. on Automated Softw. Eng. (Keynote)*. Essen, Germany: ACM, 2012.

[40] D. R. White, J. Clark, J. Jacob, and S. M. Poulding, "Searching for resource-efficient programs: Low-power pseudorandom number generators," in *Proc. GECCO*. ACM, 2008, pp. 1775–1782.

[41] D. Pesch, S. Rea, V. Zavrel, J. Hensen, D. Grimes, B. O'Sullivan *et al.*, "Globally optimised energyefficient datacenters," *ICT-Energy Concepts for Energy Efficiency and Sustainability*, 2017.