Coping with Resource Fluctuations: The Run-time Reconfigurable Functional Unit Row Classifier Architecture

Tobias Knieper[1], Paul Kaufmann[1], Kyrre Glette[2], Marco Platzner[1], and Jim Torresen[2]

[1] University of Paderborn, Department of Computer Science, Warburger Str. 100, 33098 Paderborn, Germany
{tknieper, paul.kaufmann, platzner}@upb.de

[2] University of Oslo, Department of Informatics, P.O. Box 1080 Blindern, 0316 Oslo, Norway,
{kyrrehg, jimtoer}@ifi.uio.no

Abstract. The evolvable hardware paradigm facilitates the construction of autonomous systems that can adapt to environmental changes and degrading effects in the computational resources. Extending these scenarios, we study the capability of evolvable hardware classifiers to adapt to intentional run-time fluctuations in the available resources, i.e., chip area, in this work. To that end, we leverage the Functional Unit Row (FUR) architecture, a coarse-grained reconfigurable classifier, and apply it to two medical benchmarks, the Pima and Thyroid data sets from the UCI Machine Learning Repository. We show that FUR's classification performance remains high during changes of the utilized chip area and that performance drops are quickly compensated for. Additionally, we demonstrate that FUR's recovery capability benefits from extra resources.

1 Introduction

Evolvable hardware (EHW) denotes the combination of evolutionary algorithms with reconfigurable hardware technology to construct self-adaptive and self-optimizing hardware systems. The term *evolvable hardware* was coined by de Garis [1] and Higuchi [2] in 1993.

While the majority of EHW related work focus on the evolution of functional correct circuits or circuits with a high functional quality, some authors investigates the robustness of EHW. The related literature spans this area from offline evolution of fault-tolerant circuits able to withstand defects in silicon [3] without increasing circuit's size significantly [4] or compensating supply voltage drifts [5] by recurrent re-evolution after a series of deteriorating events as the wide-band temperature changes or radiation beams treatments [6,7].

Evolvable hardware has a variety of applications, one of which are classifier systems. A number of studies report on the use of EHW for classification applications such as character recognition [8], prosthetic hand control [9], sonar

return classification [10,11], and face image recognition [10]. These studies have demonstrated that EHW classifiers can outperform traditional classifiers such as artificial neural networks (ANNs) in terms of classification accuracy. For the electromyographic (EMG) signal classification, it has been showed that EHW approaches can perform close to the modern state-of-the-art classification methods such as support vector machines (SVMs) [9] .

In this work we focus on robust EHW-based classifiers. The novelty is that we investigate classifier systems able to cope with changing resources at run-time and evaluate their classification performance while changing the size of the utilized chip area. To this end, we leverage the Functional Unit Row (FUR) architecture, a scalable and run-time reconfigurable classifier architecture introduced by Glette et al. [12]. During optimization, we increase and decrease the number of pattern matching elements included in FUR and study the development of the resulting classification accuracy and, specifically, the recovery capability of FUR.

In contrast to most previous work that studies self-adaptation in response to stimuli from outside the system, we explicitly build our analysis on the assumption of resource competition between different tasks run inside an adaptable system.

The paper is structured as follows: Section 2 presents the FUR architecture for classification tasks, its reconfigurable variant and the applied evolutionary optimization method. Benchmarks together with an overfitting analysis as well as the experiments with the reconfigurable FUR architecture are shown in Section 3. Section 4 concludes the paper and gives an outlook on future work.

2 The Reconfigurable Functional Unit Row Architecture

The Functional Unit Row (FUR) architecture for classification tasks was first presented by Glette in [12]. It is an architecture tailored to online evolution combined with fast reconfiguration. To facilitate online evolution, the classifier architecture is implemented as a circuit whose behavior and connections can be controlled through configuration registers, similar to the approach of Sekanina [7]. By writing the genome bitstream produced by a GA to these registers, one obtains the phenotype circuit which can then be evaluated. In [13], it was shown that the partial reconfiguration capabilities of FPGAs can be used to change the architecture's footprint. The amenability of FUR to partial reconfiguration is an important precondition for our work. In the following, we present the organization of the FUR architecture, the principle of the reconfigurable FUR architecture, and the applied evolutionary technique. For details about the implementation of FUR we refer to [12].

2.1 Organization of the FUR Architecture

Fig. 1 shows the overall organization of the FUR architecture. The FUR architecture is rather generic and can be used together with different basic pattern

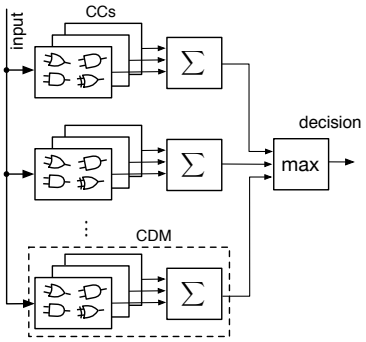

Fig. 1: The Functional Unit Row (FUR) Architecture is hierarchically partitioned for every category into Category Detection Modules (CDMs). For an input vector, a CDM calculates the likeliness for a previously trained category by summing up positive answers from basic pattern matching elements: the Category Classifiers (CCs). The CDM with most activated CCs defines the FUR's decision.

matching primitives [9,10]. It combines multiple pattern matching elements into a single module with graded output detecting one specific category. A majority voter decides for a specific category by identifying the module with the highest number of activated pattern matching elements. More specifically, for C categories the FUR architecture consists of C Category Detection Modules (CDMs). A majority vote on the outputs of the CDMs defines the FUR architecture decision. In case of a tie, the CDM with the lower index wins. Each CDM contains M Category Classifiers (CCs), basic pattern matching elements evolved from different randomly initialized configurations and trained to detect CDM's category. A CDM counts the number of activated CCs for a given input vector, thus the CDM output varies between 0 and M.

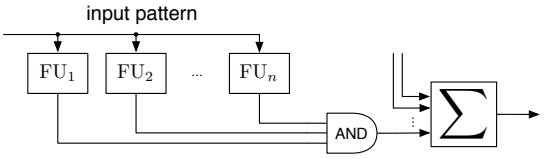

Fig. 2: Category Classifier (CC): n Functional Units (FUs) are connected to an n-input AND gate. Multiple CCs with a subsequent counter for activated CCs define a CDM.

The architecture becomes specific with the implementation of the CCs. In our case we define a single CC as a row of Functional Units (FUs), shown in Fig. 2. The FU outputs are connected to an AND gate such that in order for a CC to be activated all FU outputs have to be 1. Each FU row is evolved from

an initial random bitstream, which ensures a variation in the evolved CCs. The number of FU rows defines the resolution of the corresponding CDM.

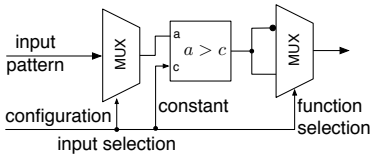

Fig. 3: Functional Unit (FU): The data MUX selects which of the input data to feed to the functions "$>$" and "$\leq$". The constant c is given by the configuration lines. Finally, a result MUX selects which of the function results to output.

The FUs are reconfigurable by writing the architecture's register elements. As depicted in Fig. 3, each FU behavior is controlled by configuration lines connected to the configuration registers. Each FU has all input bits to the system available at its inputs, but only one data element (e.g., one byte) is selected. This data is then fed to the available functions. While any number and type of functions could be imagined, Fig. 3 illustrates only two functions for clarity. In addition, the unit is configured with a constant value, c. This value and the input data element are used by the function to compute the output of the unit. Based on the data elements of the input, the functions available to the FU elements are *greater than* and *less than or equal*. Through experiments these functions have shown to work well, and intuitively this allows for discriminating signals by looking at the different amplitudes.

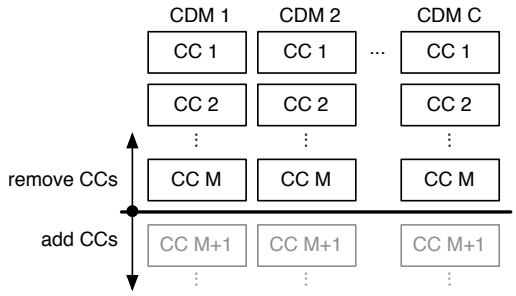

Fig. 4: Reconfigurable Functional Unit Row Architecture: The FUR architecture is configured by the number of categories, FU rows and FUs per FU row. In our work we fix the number of categories and FUs per FU rows while changing the number of FU rows per CDM.

2.2 Reconfigurable FUR Architecture

The notion of *Evolvable Hardware* bases on circuit optimization and reconfiguration. EHW-type adaptable systems improve their behavior in response to system internal and external stimuli, offering an alternative to classically engineered adaptable systems. While the adaptation to environmental changes represents the main research line within the EHW community, the ability to balance resources dynamically between multiple concurrent applications is still a rather unexplored topic. One the one hand, an EHW module might run as one out of several applications sharing a system's restricted reconfigurable resources. Depending on the current requirements, the system might decide to switch between multiple applications or run them concurrently, albeit with reduced logic footprints and reduced performance. We are interested in scalable EHW modules and architectures that can cope with such changing resource profiles. On the other hand, the ability to deal with fluctuating resources can be used to support the optimization process, for example by assigning more resources when the speed of adaptation is crucial.

The FUR architecture precisely fits this requirement as its structure can be changed (disregarding the register-reconfigurable FUs) along three dimensions, namely the number of

- categories,
- FU rows in a category, and
- FUs in a FU row.

In this work we assume the numbers of categories and FUs in a FU row as constants reconfiguring the number of FU rows in a CDM. This is illustrated in Fig. 4. For a sequence $I = \{i_1, i_2, \ldots, i_k\}$ we evolve a FUR architecture having i_1 FUs per CDM, then switching to i_2 FUs per CDM and re-evolving the architecture without flushing the configuration evolved so far. The key insights we want to gain by this investigation are the sensitivity of the FUR architecture measured in the classification accuracy to changes in the resources and the time for re-establishing near asymptotic accuracy quality.

2.3 Evolution of FUR Architecture

To evolve a FUR classifier we employ a $1 + 4$ ES scheme. In contrast to previous work [12], we do not use incremental evolution evolving CDMs and FU rows separately but evolve the complete FUR architecture in a single ES run. The mutation operator is configured to mutate three genes in every FU row.

In preparation to the experiments on the reconfigurable FUR architecture we investigate FUR's general performance by evaluating it on a set of useful FU rows per CDM and FUs per FU row configurations. The performance is calculated by a 12-fold Cross Validation (CV) scheme.

3 Experiments and Results

In this section we present two kinds of results. Initially, we analyze FURs behavior by successively testing a range of parameter combinations. Combined with an overfitting analysis we are then able to picture FUR's complete behavior for a given benchmark. Afterwards, we select a good-performing configuration to investigate FUR's performance, when being reconfigured during run-time. For this experiment we define multiple FUR architecture configurations with varying number of FU rows and plot the accuracy development, when switching between the configurations.

3.1 Benchmarks

For our investigations we rely on the UCI machine learning repository [14] and specifically, on the Pima and the Thyroid benchmarks. Pima, or the *Pima Indians Diabetes* data set is collected by the John Hopkins University in Baltimore, MD, USA and consists of 768 samples with eight feature values each, divided into a class of 500 samples representing negative tested individuals and a class of 268 samples representing positive tested individuals.

The data of the Thyroid benchmark represents samples of regular individuals and individuals suffering hypo- and hyperthyroidism. Thus, the samples are divided into 6.666, 166 and 368 samples representing regular, subnormal and hyper-function individuals. A sample consists of 22 feature values.

The Pima and the Thyroid benchmarks don't rely on high classification speeds of EHW hardware classifiers, however, these benchmarks have been selected because of their pronounced effects in the run-time reconfiguration experiment revealing FUR's characteristics.

3.2 Accuracy and Overfitting Analyses

We implement FUR's parameter analysis by a grid search over the number of FU rows and number of FUs. For a single (i, j)-tuple, where i denotes the number of FU rows and j the number of FUs, we evolve a FUR classifier by running the evolutionary algorithm for 100.000 generations. As we employ a 12-fold cross validation scheme, the evolution is repeated 12 times while alternating the training and test data sets. During the evolution we log for every increase in the training accuracy FUR's performance on the test data set. The test accuracies are not used while the evolution runs. To detect the test accuracy where the FUR architecture starts to approximate the training set tightly and to contemporary lose its ability to generalize, we average the test accuracies logged during the evolutionary runs and select the termination training accuracy according to the highest average test accuracy. This is shown in Fig. 5 for the Pima benchmark and the $(30, 8)$ configuration. The test accuracy, drawn along the y-axis, rises in relation to the training accuracy, drawn along the x-axis, until the training

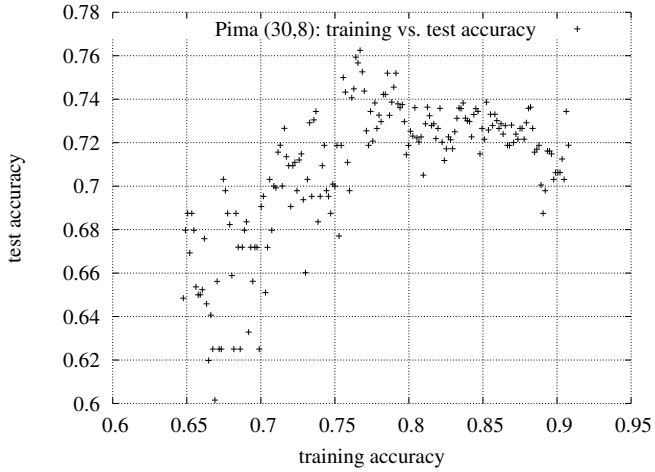

Fig. 5: Overfitting analysis: In this example the test and training accuracies would be roughly 0.76 and 0.76, respectively.

accuracy reaches 0.76. After this point the test accuracy degrades gradually. Consequently, we note 0.76 and 0.76 as the best combination of test and termination training accuracies.

To cover the interesting parameter areas and keep the computational effort low we evaluate the Pima and Thyroid benchmarks for $2, 4, 6, \ldots, 20$ FUs per FU row and for $2, 4, 6, 8, 10, 14, 16, 20, 25, 30, 35, 40, 50, 60, 70, 80$ FU rows. Fig. 6 shows the results for both benchmarks. In the horizontal level the diagrams span the parameter area of FU rows and FUs. The accuracy for each parameter tuple is drawn along the z-axis with a projection of equipotential accuracy lines on the horizontal level. While the test accuracies for the Pima benchmark, presented in Fig. 6(a) are largely independent from the number of FUs and FU rows with small islands of improved behavior around the $(8, 8-10)$ configurations, the Thyroid benchmark presented in Fig. 6(c) has an performance loss in regions with a large number of FUs and few FU rows.

Tables 1 and 2 compare FUR's results for the Pima and the Thyroid benchmarks to related work. Additionally, we use the data mining tool RapidMiner [15] to create numbers for standard and state-of-the-art algorithms and their modern implementations. To this, we evaluate in a 12-fold cross validation manner the algorithms: Decision Trees (DTs), k-th Nearest Neighbor (kNN), Multi-layer Perceptrons (MLPs), Linear Discriminant Analysis (LDA), Support Vector Machines (SVMs) and Classification and Regression Trees (CART). For the Pima benchmark our architecture outperforms any other method. It forms together with SVMs, LDA, Shared Kernel Models and kNNs a group of best performing algorithms within a 3% margin. The accuracy range of the Thyroid-benchmark is much smaller because of the irregular category data size proportions and a single dominant category amounting for 92.5% of the data. In this benchmark our architecture lies 0.66% behind the best algorithm.

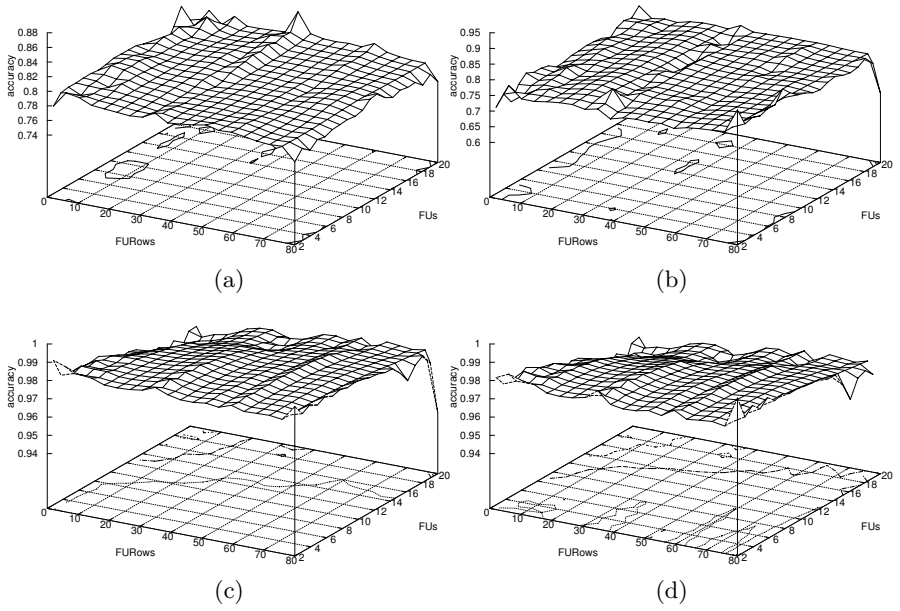

Fig. 6: Pima and Thyroid overfitting analysis: Best generalization and the according termination training accuracies for the Pima (a) (b) and the Thyroid (c) (d) benchmarks, respectively.

3.3 Reconfigurable FUR Architecture Results

In our second experiment we investigate the question of FUR classification behavior under changes in the available resources while being under optimization. We execute for both benchmarks a single experiment where we configure a FUR architecture with 4 FUs per FU row and change the number of FUs every 40.000 generations. We split the data set into disjoint training and test sets analog to the previously used 12-fold cross validation scheme and start the training of the FUR classifier with 40 FU rows. Then, we gradually change the number of employed FU rows to 38, 20, 4, 3, 2, 1, 20, 30, 40 executing altogether 400.000 generations. Fig. 7 shows the results for the Pima benchmark. We observe the following:

- The training accuracy drops significantly for almost any positive and negative change in the number of FU rows and recovers subsequently.
- While the asymptotic training accuracy is lower when using only few FU rows, the test accuracy tends to reach for any FU row configuration the usual accuracy rate. This behavior is visible from generation 120.000 to 280.000 in Fig. 7 and is confirmed by previous results showed in Fig. 6 (a).
- The recovery rate of the test accuracy depends on the amount of FU rows. While for periods with few FU rows the recovery rate is slow, for periods with 20 and more FU rows the evolutionary process manages to recover the test accuracy much faster. Interestingly, the rise of the training accuracy for

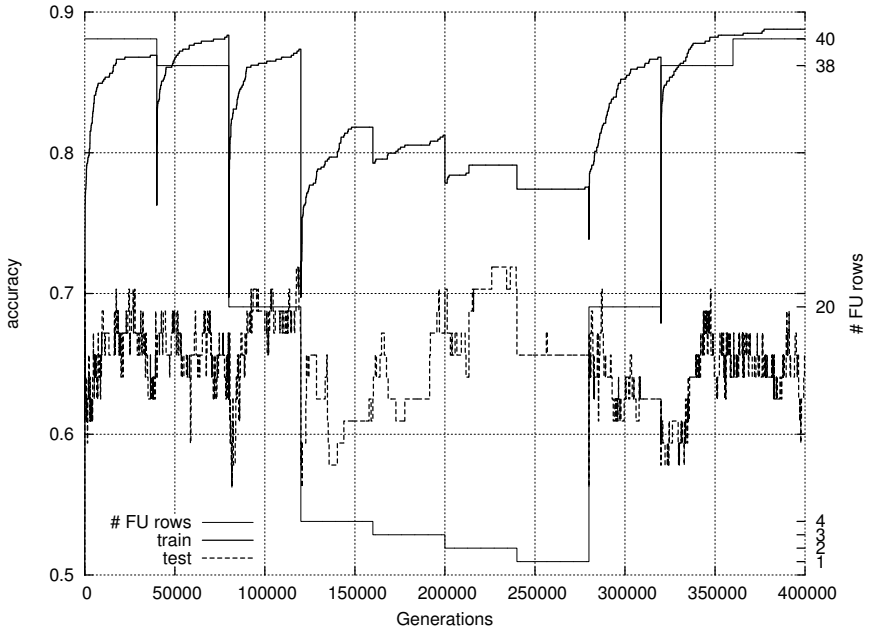

Fig. 7: The Reconfigurable Pima benchmark: Changing classifier's resources (number of FU rows) during the optimization run.

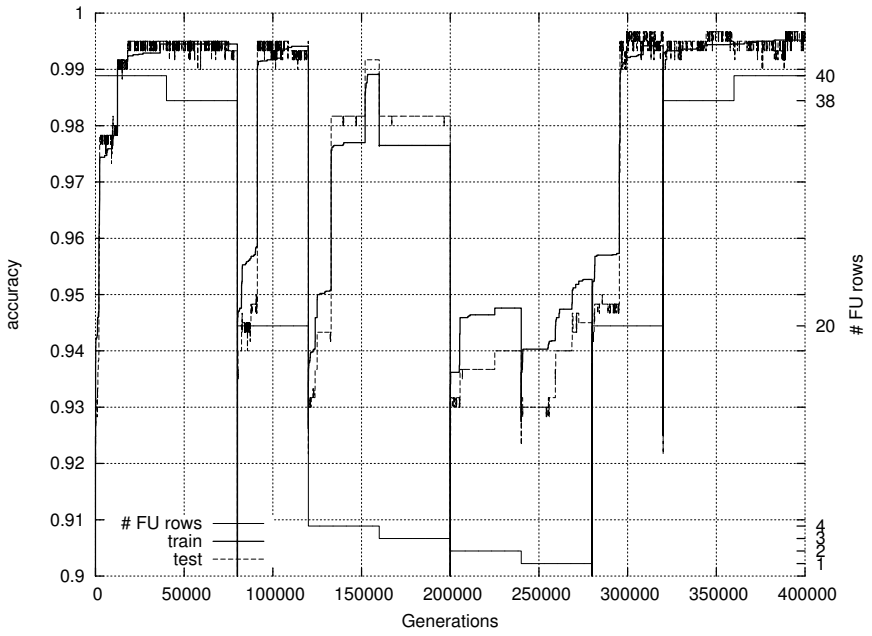

Fig. 8: Reconfigurable Thyroid benchmark: Changing classifier's resources (number of FU rows) during the optimization run.

Algorithm	Error Rate	± Standard Deviation
FUR	**21.35**	
SVM*	22.79	4.84
LDA*	23.18	4.64
Shared Kernel Models	23.27	2.56
kNN*	23.56	3.07
GP with OS, |pop|=1.000	24.47	3.69
CART*	25.00	3.61
DT*	25.13	4.30
GP with OS, |pop|=100	25.13	4.95
MLP*	25.26	4.50
Enhanced GP	25.80 – 24.20	
Simple GP	26.30	
ANN	26.41 – 22.59	1.91 – 2.26
EP / kNN	27.10	
Enhanced GP (Eggermont et al.)	27.70 – 25.90	
GP	27.85 – 23.09	1.29 – 1.49
GA / kNN	29.60	
GP (de Falco et al.)	30.36 – 24.84	0.29 – 1.30
Bayes	33.40	

Table 1: Pima benchmark: Error rates and standard deviation in %. We use the data mining toolbox RapidMiner [15] to evaluate the algorithms marked by "*". Preliminary, we identify good performing algorithm parameters by a grid search. Remaining results are taken from [16].

generations 280.000 to 320.000 results in a falling test accuracy. This could be a statistical effect, where the test accuracy varies in some interval as the classifier is evolved from a random initialized configuration.
– The test accuracy is mostly located between 0.6 and 0.7, independent of the changes in the number of FU rows. Thus, and this is the main observation, the FUR architecture shows to a large extent a robust test accuracy behavior under reconfiguration for the Pima benchmark.

Figure 8 presents the results for the Thyroid benchmark. We observe the following:

– The training accuracy, similar to the Pima results, drops significantly when changing the number of FU rows.
– As anticipated by previous results showed in Fig. 6 (c), the test accuracy drops for FUR architecture configurations with very few FU rows. This can be observed in Fig. 8 at generations 120.000 to 280.000.
– Because of the uneven distribution of category data sizes the test accuracy deviation is smaller and follows more tightly the development of the training accuracy.
– Analog to the observations made by the Pima benchmark, more FU rows increase the test accuracy recovery rate.

Algorithm	Error Rate	± Standard Deviation
DT*	0.29	0.18
CART*	0.42	0.27
CART	0.64	
PVM	0.67	
Logical Rules	0.70	
FUR	**1.03**	
GP with OS	1.24	
GP	1.44 – 0.89	
BP + local adapt. rates	1.50	
ANN	1.52	
BP + genetic opt.	1.60	
GP	1.60 – 0.73	
Quickprop	1.70	
RPROP	2.00	
GP (Gathercole et al.)	2.29 – 1.36	
SVM*	2.35	0.51
MLP*	2.38	0.62
ANN	2.38 – 1.81	
PGPC	2.74	
GP (Brameier et al.)	5.10 – 1.80	
kNN*	5.96	0.44

Table 2: Thyroid benchmark: Error rates and standard deviation in %. We use the data mining toolbox RapidMiner [15] to evaluate the algorithms marked by "*". Preliminary, we identify good performing algorithm parameters by a grid search. Remaining results are taken from [16].

– The main result is that reconfigurations of the FUR architecture are quickly compensated in the test accuracy. The limitation in the case of the Thyroid benchmark is a minimum amount of FU rows to leverage robust behavior.

In summary, as long as the FUR configuration contains enough FU rows, FUR's test accuracy behavior is stable during reconfigurations. Additionally, more FU rows leverage faster convergence.

4 Conclusion

In this work we propose to leverage the FUR classifier architecture for creating evolvable hardware systems that can cope with fluctuating resources. We describe this reconfigurable FUR architecture and experimentally evaluate it on two medical benchmarks. First, we analyze the overfitting behavior and show that the FUR architecture performs similar or better than state-of-the-art classification algorithms. Then we demonstrate that FUR's generalization performance is robust to changes in the available resources as long as a certain amount of FU rows is present in the system. Furthermore, FUR's capability to recover from a change in the available resources benefits from additional FU rows.

References

1. de Garis, H.: Evolvable Hardware: Genetic Programming of a Darwin Machine. In: Intl. Conf. of Artificial Neural Nets and Genetic Algorithms. Springer (1993) 441–449
2. Higuchi, T., Niwa, T., Tanaka, T., Iba, H., de Garis, H., Furuya, T.: Evolving Hardware with Genetic Learning: a First Step Towards Building a Darwin Machine. In: From Animals to Animats, MIT Press (1993) 417–424
3. Miller, J., Hartmann, M.: Untidy Evolution: Evolving Messy Gates for Fault Tolerance. In: Intl. Conf. of Evolvable Systems (ICES). Volume 2210 of LNCS., Springer (2001) 14–25
4. Haddow, P.C., Hartmann, M., Djupdal, A.: Addressing the Metric Challenge: Evolved versus Traditional Fault Tolerant Circuits. In: Adaptive Hardware and Systems (AHS), IEEE (2007) 431–438
5. Sekanina, L.: Evolutionary Design of Gate-Level Polymorphic Digital Circuits. In: EvoWorkshops. Volume 3449 of LNCS., Springer (2005) 185–194
6. Stoica, A., Zebulum, R.S., Keymeulen, D., Daud, T.: Transistor-Level Circuit Experiments Using Evolvable Hardware. In: Artificial Intelligence and Knowledge Engineering Applications (IWAC'05). Volume 3562 of LNCS., Springer (2005) 366–375
7. Sekanina, L.: Evolutionary Functional Recovery in Virtual Reconfigurable Circuits. Journal of Emerging Technologies in Computing Systems **3**(2) (2007)
8. Higuchi, T., Iwata, M., Kajitani, I., Iba, H., Hirao, Y., Manderick, B., Furuya, T.: Evolvable Hardware and its Applications to Pattern Recognition and Fault-Tolerant Systems. In: Towards Evolvable Hardware: The evolutionary Engineering Approach. Volume 1062 of LNCS. Springer (1996) 118–135
9. Glette, K., Gruber, T., Kaufmann, P., Torresen, J., Sick, B., Platzner, M.: Comparing Evolvable Hardware to Conventional Classifiers for Electromyographic Prosthetic Hand Control. In: Adaptive Hardware and Systems (AHS), IEEE (2008) 32–39
10. Yasunaga, M., Nakamura, T., Yoshihara, I.: Evolvable Sonar Spectrum Discrimination Chip Designed by Genetic Algorithm. In: Systems, Man and Cybernetics. Volume 5., IEEE (1999) 585–590
11. Glette, K., Torresen, J.: A Flexible On-Chip Evolution System Implemented on a Xilinx Virtex-II Pro Device. In: Intl. Conf. of Evolvable Systems (ICES). Volume 3637 of LNCS., Springer (2005) 66–75
12. Glette, K., Torresen, J., Yasunaga, M.: An Online EHW Pattern Recognition System Applied to Face Image Recognition. In: Applications of Evolutionary Computing (EvoWorkshops). Volume 4448 of LNCS. Springer (2007) 271–280
13. Torresen, J., Senland, G., Glette, K.: Partial reconfiguration applied in an on-line evolvable pattern recognition system. In: NORCHIP 2008, IEEE (2008) 61–64
14. Asuncion, A., Newman, D.: UCI Machine Learning Repository. University of California, Irvine, School of Information and Computer Sciences (2007)
15. Mierswa, I., Wurst, M., Klinkenberg, R., Scholz, M., Euler, T.: YALE: Rapid Prototyping for Complex Data Mining Tasks. In: Intl. Conf. on Knowledge Discovery and Data Mining (KDD). (2006) 935 – 940
16. Winkler, S.M., Affenzeller, M., Wagner, S.: Using Enhanced Genetic Programming Techniques for Evolving Classifiers in the Context of Medical Diagnosis. In: Genetic Programming and Evolvable Machines. Volume 10(2)., Kluwer Academic Publishers (2009) 111–140