2014 IEEE Congress on Evolutionary Computation (CEC)
July 6-11, 2014, Beijing, China

GP-Based Kernel Evolution for L_2-Regularization Networks

Simone Scardapane, Danilo Comminiello, Michele Scarpiniti and Aurelio Uncini

***Abstract*—In kernel-based learning methods, a crucial design parameter is given by the choice of the kernel function to be used. Although there is, in theory, an infinite range of potential candidates, a handful of kernels covers the majority of actual applications. Partly, this is due to the difficulty of choosing an optimal kernel function in absence of *a-priori* information. In this respect, Genetic Programming (GP) techniques have shown interesting capabilities of learning non-trivial kernel functions that outperform commonly used ones. However, experiments have been restricted to the use of Support Vector Machines (SVMs), and have not addressed some problems that are specific to GP implementations, such as diversity maintenance. In these respects, the aim of this paper is twofold. First, we present a customized GP-based kernel search method that we apply using an L_2-Regularization Network as the base learning algorithm. Second, we investigate the problem of diversity maintenance in the context of kernel evolution, and test an adaptive criterion for maintaining it in our algorithm. For the former point, experiments show a gain in accuracy for our method against fine-tuned standard kernels. For the latter, we show that diversity is decreasing critically fast during the GP iterations, but this decrease does not seems to affect performance of the algorithm.**

I. Introduction

NOWADAYS, kernels are one of the most fundamental tools in machine learning applications. Their use in the field first saw the light in 1964, when Aizerman et al. [1] employed them in a famous convergence proof. Another important milestone was in 1992, when Boser et al. [2] derived the nonlinear Support Vector Machine (SVM) algorithm. Since then, positive-semidefinite (PSD) kernels have seen a constant growth in importance, as depicted for example by the widespread use of support vectors based algorithms [3]. The kernel evaluation can be seen as performing a dot product in a transformed input space, hence its design is a critical choice whilst crafting a specific learning system. In practice, however, a small set of kernels is used in the majority of applications, due to their simplicity and universal approximation capabilities [3], [4].

Clearly, such common kernels may result in sub-optimal performance with respect to other, non-trivial choices. As an example, there is agreement that combinations of base kernels (the so-called *Multiple Kernel Learning* framework) can result in an increased performance in kernel methods [5]. In this respect, the choice of an optimal kernel can be seen as an optimization problem, where the search space is the space $\mathcal{K}$ of all possible PSD kernels, and the target function is defined as a measure of error on the data, e.g. the classification accuracy. Due to the richness of the algebra defined on $\mathcal{K}$, however, a thorough search is by all means intractable. Genetic Programming (GP) techniques were employed successfully for similar problems [6], and for this reason over the last years several authors have proposed them for the evolution of an optimal kernel function [7]–[10]. GP algorithms construct solutions to an optimization problem incrementally, by drawing inspiration to the biological mechanism of natural selection. Moreover, they are designed to work whenever the solution of the problem admits a structured representation, such as a graph or a tree.

Despite the wide range of possible configurations for a GP implementation, all previously mentioned studies on GP-based kernel evolution share the same workflow during the initialization of the population, and in terms of genetic operators used for constructing a new population. Moreover, all of them use Support Vector Machines (SVMs) as the base learning algorithm. They differentiate themselves mostly on the way in which a single individual is constructed: either using the input vectors as base components [7], or by directly using a set of base kernels that are subsequently combined [8]. Although all the initial studies found a decrease of testing error using GP-based kernels on a set of UCI repositories [7], the work by Koch et al. [10] challenged some of these previously established results, showing that GP-based kernels do not result in a significant increase in performance against basic kernels whose parameters are well-tuned in the context of SVMs. Due to the flexibility of GP algorithms, however, they conclude that "*results seem promising enough to warrant further work in improving GP for support vector kernel evolution*" [10].

With respect to this, our aim in this paper is twofold. First, we investigate the performance of a GP-based kernel search method when employing a different kernel learning method, in particular an L_2-Regularization Network (RN) [11]. To our knowledge, this is the first study employing GP algorithms for kernel evolution outside SVMs classifiers. Secondly, we are interested in the role that *diversity* plays in kernel evolution. Diversity measures the heterogeneity of the population during the GP process (or, in other terms, the *variety* of individuals), and it is known that a fast drop in diversity during the early iterations of a GP algorithm can result in a poor performance of the overall procedure [12]. Up to this point, no studies have investigated this issue in the context of kernel evolution. However, we hypothesize that decrease of diversity and early convergence are both crucial problems that need to be considered. Basic kernels, such as the Gaussian one, can easily have a very high fitness from the start of the GP process. Hence, there is the possibility that they overcrowd the population in the first iterations, possibly shadowing better solutions that require more time

Authors are with the Department of Information Engineering, Electronics and Telecommunications (DIET), "Sapienza" University of Rome, Via Eudossiana 18, 00184, Rome. Email: {simone.scardapane, michele.scarpiniti, danilo.comminiello, aurelio.uncini }@uniroma1.it.

to be found by the procedure. This is in line with many experimental findings, e.g. [10], where the GP algorithm resulted in simply "rediscovering" basic kernels. If our hypothesis is correct, we are also interested in investigating whether adaptively maintaining a high level of diversity can increase performance.

As a starting point, we propose a customized GP procedure employing some modifications with respect to the standard one. First, as stated before, we employ a faster training method as base learning algorithm, which enable us to consider longer evolutions in the GP cycle. Secondly, we designed a different initialization method, where more importance is given to small kernels. We found that this method counteract the "random search" behavior found by Koch et al. [10], allowing the algorithm to construct new solutions incrementally. Finally, we include a post-processing phase for fine-tuning the regularization parameter. Results show that our algorithm has a significant increase in performance on the same UCI datasets considered by previous works, even against standard kernels whose parameters are finely tuned by a grid search procedure. Answers to the second question, however, are less clear. We show that our hypothesis is correct, and diversity indeed decreases critically fast, under many different control measures. Maintaining it, however, seems to have only a slight increase in performance. Therefore, we think this is an interesting starting point for further research on the problem.

The rest of the paper is organized as follows. In Section II we give a brief overview of the kernel method that we employ, L_2-Regularization Networks. Next, in Section III we present our GP algorithm, with a particular emphasis on the novel aspects that are introduced. In Section IV we describe three measures that we use to analyze the evolution of diversity, and we detail a simple adaptive criterion for maintaining variety in the GP evolution. Finally, Section V presents and analyzes the experimental results.

II. Regularization Networks

Consider the standard supervised learning setting [11], whose objective is finding a multivariate function that approximates as best as possible an unknown relation between an input space $\mathcal{X} \subseteq \mathbb{R}^D$ and an output space $\mathcal{Y}$. The output space changes depending on the task we are considering. Typically, in classification we have $\mathcal{Y} = \{1, \ldots, M\}$, where M is the number of classes, while in (one-dimensional) regression we have $\mathcal{Y} \subseteq \mathbb{R}$. We are given a set of N examples of this relation, in the form of a dataset $\{\mathbf{x}_i, y_i\}_{i=1}^{N}, \mathbf{x}_i \in \mathcal{X}$, $y_i \in \mathcal{Y}$.

Regularization theory [11] formulates the problem as the one of finding the function f that minimizes the following functional:

$$\min_{f \in \mathcal{H}} \frac{1}{N} \sum_{i=1}^{N} L(y_i, f(\mathbf{x}_i)) + \lambda \|f\|_{\mathcal{H}}^2 \tag{1}$$

where $\mathcal{H}$ is a (normed) hypothesis space, $L(\cdot, \cdot)$ a loss function giving the cost we are incurring for any wrong prediction, and $\lambda \geq 0$ is a scalar, known as regularization factor, that balances the two terms. If $\mathcal{H}$ is a *Reproducing Kernel Hilbert Space* (RKHS), the Representer's Theorem [11] asserts that solutions to Eq. (1) have the form:

$$f(\mathbf{x}) = \sum_{i=1}^{N} \alpha_i k(\mathbf{x}, \mathbf{x}_i) \tag{2}$$

where $k(\cdot, \cdot)$ is the kernel associated to the RKHS, and $\alpha_i, i = 1, \ldots, N$ a set of real scalars. The condition that $\mathcal{H}$ is a RKHS can be equivalently expressed in terms of the kernel, by defining the *kernel matrix* $\mathbf{K} : K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$. It can be shown that Eq. (2) holds if and only if, for any possible choice of the dataset and any $\mathbf{z} \in \mathcal{X}$, we have:

$$\mathbf{z}^T \mathbf{K} \mathbf{z} \geq 0$$

i.e., $\mathbf{K}$ is positive semi-definite (PSD). In this case we also say that the kernel function is PSD. In this framework, Support Vector Machines for binary classification (C-SVM) are obtained by using the Hinge loss function $L(y_i, f(\mathbf{x}_i)) = (1 - y_i f(\mathbf{x}_i))_+$, where $(a)_+ = \max\{0, a\}$. This results in a quadratic programming problem. A simpler optimization problem can be obtained by considering the squared loss function $L(y_i, f(\mathbf{x}_i)) = (y_i - f(\mathbf{x}_i))^2$. In this case the resulting model is called an (L_2) Regularization Network (RN), and the coefficients of Eq. (2) are obtained by solving the following linear system of equations:

$$(\mathbf{K} + \lambda \mathbf{I}_N)\boldsymbol{\alpha} = \mathbf{y} \tag{3}$$

where we defined $(\boldsymbol{\alpha})_i = \alpha_i$, $(\mathbf{y})_i = y_i$ and $\mathbf{I}_N$ is the $N \times N$ identity matrix. This formulation can be extended to take into consideration a bias term [11], which is equivalent to considering a shift of the decision boundary. Eq. (3) can be solved more efficiently with respect to SVMs, providing the possibility of considering longer evolutions in the GP search. Moreover, RNs can be used directly for multi-class classification, while the standard SVM model is formulated only in terms of binary classification. Thus, a multi-class problem has to be decomposed into smaller binary classification problems, resulting in the necessity of training multiple SVMs for a single task.

III. GP-Based Kernel Evolution

In this section we describe our GP-based algorithm for evolving kernel functions. The base structure is taken from [9] and is exemplified in Fig. 1. Each individual in the GP procedure (represented as a tree) identifies a kernel function, whose fitness is computed depending on the validation error it achieves on a given dataset. Individuals are recombined for a predefined number of iterations until a termination criterion is met, and the kernel with highest fitness is returned in output to the algorithm. In the following we describe with more detail each block of Fig. 1, with a particular emphasis on the modifications we introduced.

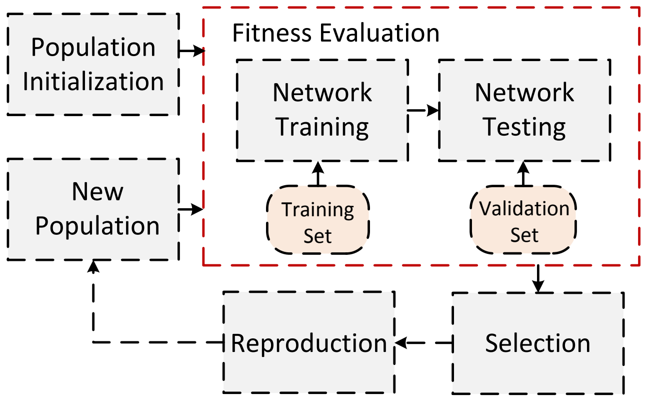


Fig. 1. Block Diagram of the Algorithm

A. Initialization

A possible kernel (denoted as an *individual*) is represented as a tree. Each leaf node of the tree can be one of four basic kernels[1]:

- Linear kernel: $k(\mathbf{x},\mathbf{y}) = \langle \mathbf{x},\mathbf{y} \rangle$, where $\langle \mathbf{x},\mathbf{y} \rangle$ is the standard inner product in $\mathbb{R}^N$ defined as $\mathbf{x}^T\mathbf{y}$.
- (Homogeneous) Polynomial kernel: $k(\mathbf{x},\mathbf{y}) = \langle \mathbf{x},\mathbf{y} \rangle^d$, where $d \in \mathbb{N}$.
- Gaussian kernel: $k(\mathbf{x},\mathbf{y}) = \exp\left\{-\gamma \left\|\mathbf{x}-\mathbf{y}\right\|_2^2\right\}$, where $\gamma \in \mathbb{R}$.
- Hyperbolic kernel: $k(\mathbf{x},\mathbf{y}) = \tanh\{\gamma \langle \mathbf{x},\mathbf{y} \rangle + r\}$, where $\gamma, r \in \mathbb{R}$.

Note that the polynomial kernel has the linear kernel as a special case (with $d = 1$), but we consider them separately to give more emphasis to the linear one during construction of an individual. An internal node of the tree can be one of the following operations:

- Sum of two kernels: $k(\mathbf{x},\mathbf{y}) = k_1(\mathbf{x},\mathbf{y}) + k_2(\mathbf{x},\mathbf{y})$,
- Scaling of a kernel: $k(\mathbf{x},\mathbf{y}) = a \times k_1(\mathbf{x},\mathbf{y})$,
- Shifting of a kernel: $k(\mathbf{x},\mathbf{y}) = k_1(\mathbf{x},\mathbf{y}) + a$,
- Kernel product: $k(\mathbf{x},\mathbf{y}) = k_1(\mathbf{x},\mathbf{y}) \times k_2(\mathbf{x},\mathbf{y})$,
- Exponentiation of a kernel: $k(\mathbf{x},\mathbf{y}) = \exp\{k_1(\mathbf{x},\mathbf{y})\}$,

where $k_1(\cdot,\cdot), k_2(\cdot,\cdot)$ can be themselves composite kernels, and $a \in \mathbb{R}$. These operations are chosen since they respect the *closure* properties of positive-semidefinite kernels [9][2]. Hence, we are assured that if we respect the n-arity of the operators while recombining individuals, the resulting kernel will always be positive-semidefinite. The algorithm is started by initializing a pool of P individuals, that is known as a *population*. Differently from previous works, we do not use the standard *grow* method [6] for initializing a kernel. Instead, we found that a better choice was to give more emphasis in the beginning to small kernels:

[1]In the following, we use the terms *basic* and *standard* interchangeably to refer to these four kernels.

[2]Shifting of a kernel is not a standard operation in this context, but it can be seen as a shift of the hyperplane in the transformed space. See [11] and the discussion in Section II.

- A third of the population is initialized using the four basic kernels detailed before. The parameters are generated randomly according to the procedure detailed in [9]. Moreover, slightly less probability is given to the linear kernel of being chosen.
- A third of the population is composed using kernels of depth 2, i.e., single operations on basic kernels. Here, internal nodes have not the same probability of being generated. In particular, addition and multiplication are more probable than scaling and shifting, which in turn are more probably that exponentiation. These rules of thumb were chosen based on a large series of experimental tests with our procedure.
- The final third of the population is composed of kernels of depth 3. Generation of internal nodes follows the same guidelines as the previous point.

A measure of fitness is evaluated for each individual, by training an RN model and then validating its performance over an independent validation set. For robustness, this evaluation is repeated K times using a K-fold cross-validation. The accuracy of each kernel up to a certain tolerance t (3 decimal places in our experiments) is then assigned as its fitness value. It can happen that individuals, even if they respect the condition of positive semi-definitiveness, result in highly ill-conditioned inversion problems in Eq. (3). In previous works [9] this had the consequence of making the SVM solver hangs indefinitely. Using RNs, we can check this by computing the 2-*norm condition number* r of the $\mathbf{K}$ matrix [13] and assigning the lowest possible fitness value whenever r is less than a certain threshold (10^{-6} in our experiments).

B. Mutation and Crossing-Over

Based on the fitness values, a new population of kernels is composed by selecting individuals and recombining them. A full cycle of fitness evaluation/recombination is called a *generation*. Selection is done through *lexicographic tournament selection*, where a set of T individuals, with T chosen *a priori*, is extracted at random from the population, and the fittest one is then selected. In case of equal fitness, the one with fewer nodes is selected. We refer to the number of nodes of an individual as its *length*.

We introduce in our algorithm an *elitist* procedure, i.e., the fittest E individuals of the previous population are always chosen for the next generation. Then, the rest of the new population is constructed as follows:

- A fraction $r(P-E)$ of individuals, with $r \in [0,1]$, are created by crossover over two selected parents. An example of this operation is detailed in Fig. 2. Each node in the two parents is labeled with an integer number using a depth-first strategy. Two nodes, one for each parent, are randomly selected and the corresponding trees are interchanged. Finally, the resulting individuals are pruned if they exceed a maximum depth of d, by substituting non-terminal nodes at depth d with one of their randomly chosen leaf nodes.

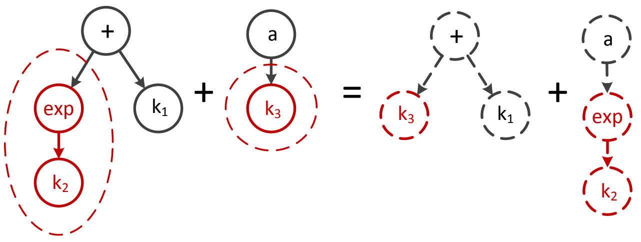


Fig. 2. Crossing-Over Between Two Kernels

TABLE I
LIST OF PARAMETERS.

Notation	Description
M	Maximum number of generations.
P	Size of the population (number of kernels).
T	Tournament size.
r	Reproduction rate (fraction of individuals created by crossover).
m	Mutation probability.
d	Maximum kernel depth.
E	Elitism degree.
K	Number of folds in K-fold cross-validation.
t	Tolerance for evaluating the fitness function.

- The remaining $(1-r)(P-E)$ individuals are selected and passed to the new population without modifications.
- Each individual in the new population has a probability $m \in [0,1]$ of being randomly mutated at one of its nodes. Mutation can happen at any node, and the new subtree is generated randomly according to the same specifications detailed in Section III-A. Moreover, depth of the mutated subtree cannot exceed depth of the original one.

As a termination criterion, the algorithm is allowed to run for a maximum of M generations. The full list of parameters is collected for simplicity in Table I. As in previous works [9], the computation complexity of our algorithm is in the order of $\mathcal{O}(MP)$ fitness evaluations. Each fitness evaluation requires the inversion of an $N \times N$ matrix, where N is the number of training samples. However in practice, as we stated in Section II, solving Eq. (3) is found to be much faster than training classical SVMs, which results in a large improvement in computational time.

C. Regularization Parameter

A critical choice in the GP evolution is given by the regularization parameter λ in Eq. (1). Increasing λ increases the regularization term, which after a certain point worsen the obtained solution. The optimal λ, however, depends on the specific kernel that is used and on the dataset itself, so it must be chosen wisely for every problem at hand. Previous works used either a fixed λ for each problem [7], or an adaptive one computed in terms of the variance of the data in the transformed space [10]. In our algorithm both approaches were found unsatisfactory, and we adopted a different method. Two RNs using a Gaussian and a Polynomial kernel are trained, and their parameters are fine-tuned using a grid search (see Section V for details on the grid search). After this, we obtain two optimal parameters, denoted as λ_P^* and λ_G^*. The regularization parameter of the GP algorithm is then initialized as $\lambda^* = \max\left\{\lambda_P^*, \lambda_G^*\right\}$. For an additional fine-tuning, after the GP evolution is over, a post-processing phase is performed by running a grid search for λ over the set $\left\{2^{-5}, \ldots, 2^{10}\right\}$ using the final kernel returned by the GP procedure.

IV. Diversity in Kernel Evolution

A. Computing Diversity

Diversity in a GP algorithm relates to the variety of individuals in a population. If a few of them compose the overall population, we say that diversity is low, and *vice versa*. It is known that a fast decrease in diversity in the early iterations of the GP procedure can be correlated with poor performance [14], and premature convergence to a local minimum. Controlling diversity is not straightforward, however, because GP individuals posses an inherent topological structure, and defining an appropriate metric to compute similarity is an open problem in the general case [14].

Diversity measures are generally categorized in *genotypical* measures, that depend on the morphological structure of each individual, and *phenotypical* measures, that only relate to the fitness values. Clearly, the former are a more accurate description of the true diversity of the population, although they are more elaborate to compute. To test the evolution of diversity in our problem, we use three widely used measures that spans both categories and that we describe next.

During a single generation, after fitness is computed for each individual, fitness values of the population are subdivided into k bins, where the number of bins is designed to be slightly coarser than the accuracy t chosen before (e.g., 10^{t-1} bins). The frequency of individuals in bin k is denoted as p_k. The *phenotypic diversity* (PD) is defined as the number of non-empty bins. The *entropy* (E), instead, is a more accurate measure defined as:

$$E = -\sum_k p_k \log\left\{p_k\right\} \tag{4}$$

As a genotypical measure, we decided to compute the *edit-2 distance* (E2D) between each individual and the best solution found so far [14]. The E2D $d(p_1, p_2)$ between two individuals p_1 and p_2 is computed as follows:

1) Starting from the root node, the two individuals are superimposed. Whenever this operation is not possible (e.g., nodes with different n-arity or terminal and non-terminal nodes) empty nodes are added to the smaller tree. After this operation, both individuals have n nodes that are in bijective correspondence. We denote with $p_i^{(j)}$ the j-th node of individual i.

TABLE II
DESCRIPTION OF THE DATASETS.

ID	Name	Input features	Instances	Desired output	Task Type
G	Glass	9	214	Glass quality	Regression
I	Ionosphere	34	315	Target detection	Classification
W	Wdbc	30	569	Cancer diagnosis	Classification
YA	Yacht	6	308	Residuary resistance	Regression
YE	Yeast	8	1484	Localization site of protein	Classification

2) The difference d_k at node k is 1 if the two nodes are of the same category (i.e., both scaling operators) and 0 otherwise. The difference between a non-empty node and an empty one is defined to be 0.
3) The E2D at node k is then computed recursively as:

$$d(p_1^{(k)}, p_2^{(k)}) = d_k + \frac{1}{2} \sum_{i \in \mathcal{N}_k} d(p_1^{(i)}, p_2^{(i)}) \qquad (5)$$

where $\mathcal{N}_k$ denotes the possibly empty set of children of node k. The E2D of two individuals is the E2D between their root nodes.

The E2D is chosen in GP algorithms because it gives more emphasis to nodes closer to the root node (due to the scaling factor in Eq. (5)). This is desired since a large body of practical findings seems to imply that these nodes have more importance in the performance of an individual [14]. Moreover, the E2D only uses information on the topology of the individual, and not on the contents of each node, which makes it more robust to small mutations of an individual. The main drawback of the E2D is that it requires an artificial ordering of the child of a node in step 1 of the computation. However, discarding this assumption would make its computation exponential in the depth of the tree, hence intractable.

B. Maintaining Diversity

The diversity measures described in the previous section can be used in the GP procedure for enforcing a high degree of variety in the individuals. Here we elaborate on a simple adaptive criterion based on a *fitness sharing* technique described in [12] to attain this objective. The main idea of fitness sharing is to change the fitness of an individual depending on its similarity with the rest of the population. This operation is controlled by a parameter, generally called *sharing factor*, that balances between the degree at which diversity is enforced and performance of the algorithm. Following [12], here we consider an adaptive sharing factor whose evolution depends on the performance of the algorithm and on the overall level of diversity of the population.

First, the E2D between each individual in the population is computed. Then, the *sharing level* s_{ij} between individual i and j is defined as:

$$s_{ij} = \begin{cases} 1 - \left(\frac{d(p_i,p_j)}{\alpha}\right)^2 & \text{if } d(p_i,p_j) \le \alpha \\ 0 & \text{otherwise} \end{cases} \qquad (6)$$

where the parameter α is called the *sharing factor*. The *total sharing* s_i of individual i is then given by:

$$s_i = \sum_{j=1}^{P} s_{ij}$$

The fitness of each individual is then normalized by its total sharing:

$$f_i' = \frac{f_i}{s_i}$$

The sharing factor is initialized to 1 and adaptively changed during the run of the algorithm every 3 generations. In particular, denote by d_E the relative change in entropy during the last 3 generations, and by d_F the relative change in average fitness. If the entropy of the population has dropped more than 50%, i.e., $d_E < 0.5$, the sharing factor is increased:

$$\alpha = \alpha / d_E$$

Conversely, if the average fitness has decreased (i.e., $d_F < 1$), the sharing factor is decreased:

$$\alpha = \alpha \cdot d_F$$

In this way, whenever diversity is decreasing rapidly, individuals which are largely different from the rest of the population have a proportionally higher fitness. However, this effect is attenuated if fitness is not increasing.

V. Experimental Results

A. Setup

We test the GP algorithm described in Section III over 5 UCI datasets taken from previous works on the subject [7], [9]. The algorithm is compared against 3 RNs using respectively (i) a Gaussian kernel (denoted as RN-G), (ii) a Polynomial kernel (denoted as RN-P), and (iii) a linear kernel (denoted as RN-L). For all three of them, the eventual parameter of the kernel and the regularization parameter are found by performing a grid search using a 3-fold cross validation as performance measure. λ and the parameters of the Gaussian kernel are searched in the interval $[2^{-5}, \ldots, 2^{10}]$, while the degree of the polynomial in $[2, \ldots, 15]$.

Regarding the RN with the GP algorithm (denoted as RN-GP), we use a population of 70 individuals which are recombined for 50 iterations. The elitism degree is set equal to 2, and the rest of the population is composed by crossover

TABLE III
EXPERIMENTAL RESULTS. MISCLASSIFICATION RATE IS SHOWN FOR DATASETS I, W AND YE, WHILE NRMSE FOR DATASETS G AND YA.

Dataset	RN-G	RN-P	RN-L	RN-GP
G	0.51	0.52	0.53	**0.49**
I	0.08	0.10	0.19	**0.05**
W	**0.02**	0.06	0.06	**0.02**
YA	0.10	0.05	0.61	**0.04**
YE	0.47	0.49	0.49	**0.45**

TABLE IV
OPTIMAL PARAMETERS FOUND BY THE GRID-SEARCH PROCEDURE.

ID	RN-G	RN-P	RN-L
G	$C = \{4, 16\}$ $\gamma = \{0.25, 0.5\}$	$C = \{1, 2\}$ $d = 2$	$C = 2$
I	$C = \{4, 8, 32\}$ $\gamma = \{0.25, 0.5\}$	$C = 0.03$ $d = \{3, 4\}$	$C = \{1, 8\}$
W	$C = \{4, 8, 16\}$ $\gamma = \{0.5, 2\}$	$C = \{4, 16, 64\}$ $d = \{1, 2\}$	$C = \{32, 256\}$
YA	$C = 1024$ $\gamma = \{0.5, 1\}$	$C = 1024$ $d = 5$	$C = \{64, 1024\}$
YE	$C = \{0.25, 1\}$ $\gamma = \{16, 64\}$	$C = \{8, 16, 32\}$ $d = \{2, 8\}$	$C = \{4, 32, 64\}$

for 90% and by selection for the remaining 10%. The mutation rate is set to 20%. Tournament size and maximum kernel depth are both 6. Finally, the fitness computation uses the same 3-fold validation defined for the grid search.

The algorithms are tested using an outer 5-fold cross validation, and results are averaged over 5 different runs. Name, number of input features and dimension of the datasets are detailed in Table II. For classification, we use the opposite of the misclassification error as the measure of performance. For regression, we use the opposite of the Normalized Root Mean-Square Error (NRMSE), defined for a testing set S as:

$$\text{NRMSE}(S) = \sqrt{\frac{\sum_{(\mathbf{x}_i, y_i) \in S} (f(\mathbf{x}_i) - y_i)^2}{|S| \hat{\sigma}_y}} \tag{7}$$

where $|S|$ denotes the cardinality of the set S and σ_y is an empirical estimate of the variance of $\mathbf{y}$. We found that, due to the presence of the normalization factor in Eq. (7), this performance measure resulted in more consistent results inside the GP procedure.

B. Results

Results of the simulations are detailed in Table III. For readability, we show the opposite of the fitness measure, i.e., misclassification rate for datasets I, W and YE, and NRMSE for datasets G and YA. The best result for each dataset is highlighted in bold. Typical parameters obtained by the grid search for the three basic RNs are in Table IV. If the grid search found different parameters in different folds, we present all the possible configurations between curly brackets. We can see from Table III that RN-GP obtains a significant decrease of error for dataset I, a slightly lower decrease for datasets G, YA and YE, and has a similar performance to the Gaussian kernel for W. Hence, the procedure seems to work consistently better than finely tuned kernels in the majority of cases under study.

It is interesting to mention that low performance, e.g. in the W dataset, seems to be correlated to an increased rate of convergence to simple kernels. As an example, in the I dataset, the average depth of the resulting kernel was 3.64, and during all the iterations (25 folds in total) the algorithm converged only three times to basic kernels. Conversely, in the W dataset, average length was 2.6, and the algorithm converged consistently to basic kernels. Consider as an example the resulting kernels for the fourth run, depicted in Table V. We see that kernels k_1 and k_2 are simple Gaussian kernels, kernels k_3 and k_5 are products of Gaussian kernels (hence Gaussian kernels) and only the kernel k_4 is a nontrivial one. It is also worth of mention that the range of the γ parameter in Table V is very similar to the optimal parameter found for RN-G using a grid search (see the fourth row of Table IV). The situation for the YA dataset (where the improvement is minimal) is in between, although we found an increased rate of convergence with respect to basic polynomial kernels of large degree.

Hence, the GP algorithm seems to oscillate between two distinct behaviors: either recovering non-trivial kernels that outperform standard ones by a few percentage points, or simply converging to them. This latter behavior can indicate an early convergence to a local minimum (a basic kernel), or simply the impossibility for the RN-GP algorithm to improve over a standard kernel. In the following subsection, we analyze this behavior from the point of view of diversity.

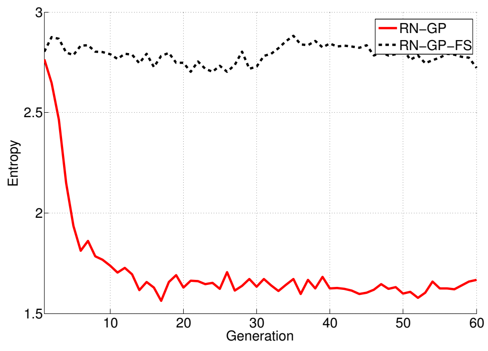
C. Analysis of Diversity Evolution

We found that diversity has a similar evolution in all the 5 datasets we investigated, resulting in a very fast decrease under the 3 measures that were detailed in Section IV. We show two representative examples, taken from datasets I and YE, in Fig. 3, where the RN-GP algorithm is shown with a solid red line. It can be seen that the average edit distance with the best individual goes in both cases to ≈ 1.4 (Fig. 3 (c)-(d)), while the final number of distinct fitness values is oscillating between 10 and 12 (Fig. 3 (e)-(f)) in the end of the runs. Similarly, entropy decreases by a factor of 3 during the first few iterations (Fig. 3 (a)-(b)). Overall, it seems that the GP procedure is converging very rapidly to a population composed of a few number of significant individuals which are then recombined, hence wasting a large part of its computational effort.

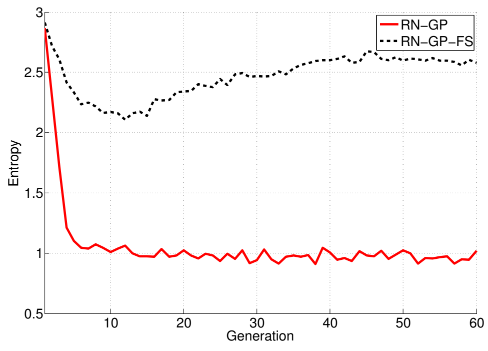
Inserting an adaptive criterion for maintaining diversity, however, did not result in a significant increase in performance. We show in Table VI the final testing error of the GP algorithm using the criterion described in Section IV-B, denoted as RN-GP-FS. The only differences with respect

TABLE V
KERNELS OBTAINED IN RUN 4 OF THE W DATASET

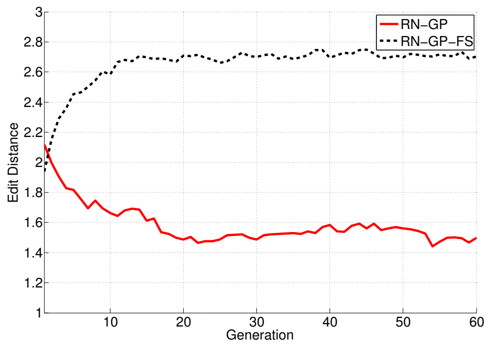
Fold	Kernel
1	$k_1(x,y) = \exp\left\{-0.9\|x-y\|^2\right\}$
2	$k_2(x,y) = \exp\left\{-0.2\|x-y\|^2\right\}$
3	$k_3(x,y) = \exp\left\{-0.2\|x-y\|^2\right\} \cdot \exp\left\{-0.6\|x-y\|^2\right\} \cdot \exp\left\{-0.4\|x-y\|^2\right\}$
4	$k_4(x,y) = 0.85 * \tanh\left\{0.0002 * \{x,y\} + 1\right\} \cdot \exp\left\{-0.2\|x-y\|^2\right\} \cdot \exp\left\{-0.6\|x-y\|^2\right\}$
5	$k_5(x,y) = \exp\left\{-0.9\|x-y\|^2\right\} \exp\left\{-0.7\|x-y\|^2\right\}$



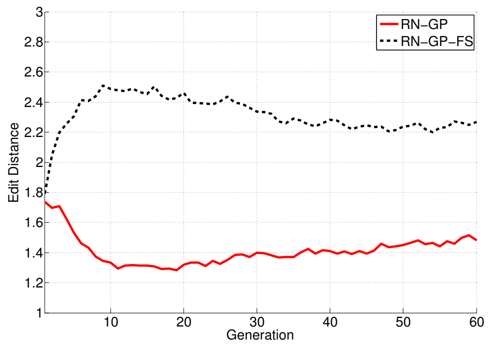
(a) Entropy for the I dataset



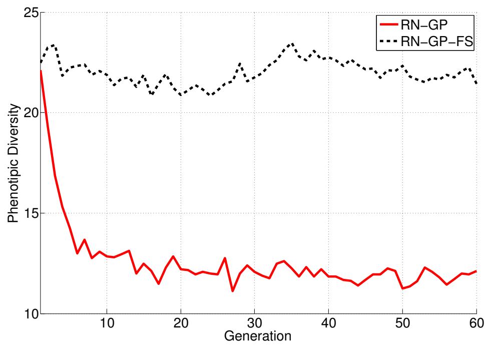
(b) Entropy for the YE dataset



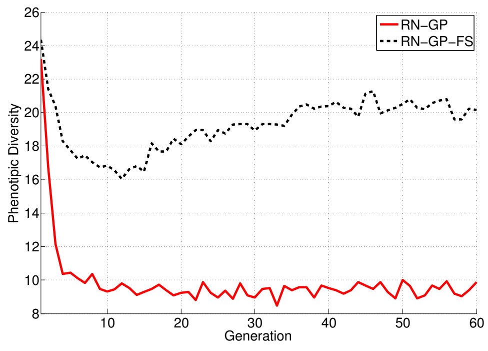
(c) Average E2D with best individual for the I dataset



(d) Average E2D with best individual for the YE dataset



(e) Phenotipic diversity for the I dataset



(f) Phenotipic diversity for the YE dataset

Fig. 3. Evolution of 3 different diversity measures for 2 representative datasets.

to the behavior of RN-GP are a small decrease of error for dataset YA, and a similar increase of testing error for dataset YE. The evolution of the different diversity measures that are computed for RN-GP-FS is shown in Fig. 3 with a dashed black line. We see that the adaptive criterion is indeed maintaining a relatively high degree of variety in the individuals, with a final edit distance with the best individual that generally goes to ≈ 2.5, and more than 20 distinct fitness values in the end. This is not compensated, however, by an equivalent increase in performance. With respect to what has been said earlier, this can be interpreted in two different ways: either the algorithm is truly converging to a global

TABLE VI
RESULTS OF ADAPTIVELY MAINTAINING DIVERSITY

Dataset	RN-GP	RN-GP-FS
G	0.49	0.49
I	0.05	0.05
W	0.02	0.02
YA	0.04	0.03
YE	0.45	0.47

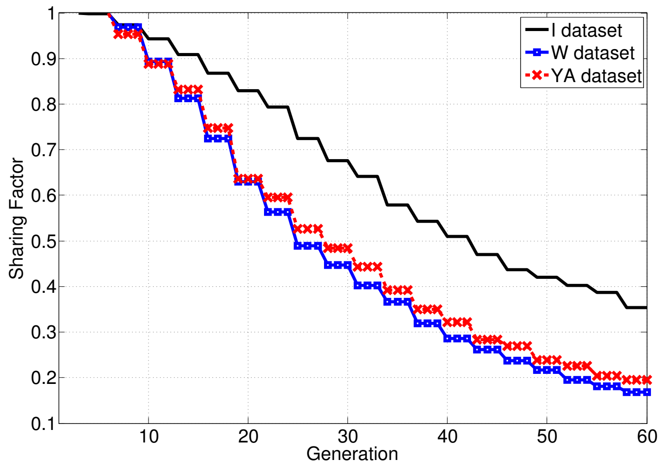


Fig. 4. Evolution of the sharing factor for 3 representative datasets.

optimum, or the adaptive criterion that we introduced is not sufficient to promote an intelligent level of diversity.

This is also exemplified by the sharing factor evolution, that we show in Fig. 4 for three representative datasets. We see that the sharing factor tends to decrease during the evolution, meaning that the algorithm is able to keep an high level of diversity with a relative ease, which is exemplified by the lack of a significant sharing factor increase in Fig. 4. Additionally, we see that promoting diversity generally results in an increase of the average fitness, which is shown by the constant decrease of the sharing factor.

VI. CONCLUSIONS

In this paper we have investigated the problem of automatically generating a kernel function by means of Genetic Programming techniques. In particular, to our knowledge this is the first work employing GP techniques for evolving the kernel of a Regularization Network. Our GP algorithm has interesting performance on a series of UCI datasets, which are consistent over the experiments we performed. We conjectured that diversity maintenance is an important factor to consider in GP evolution, and tested an adaptive criterion for enforcing it. Results show that diversity is indeed decreasing very fast, but maintaining it seems to have no consequence on the actual performance of the algorithm. Overall, we think this paves the way for further research on the topic.

REFERENCES

[1] A. Aizerman, E. M. Braverman, and L. I. Rozoner, "Theoretical foundations of the potential function method in pattern recognition learning," *Automation and remote control*, vol. 25, pp. 821–837, 1964.

[2] B. E. Boser, I. M. Guyon, and V. N. Vapnik, "A training algorithm for optimal margin classifiers," in *Proceedings of the fifth annual workshop on Computational learning theory*. ACM, 1992, pp. 144–152.

[3] T. Hofmann, B. Schölkopf, and A. J. Smola, "Kernel methods in machine learning," *The Annals of Statistics*, vol. 36, no. 3, pp. 1171–1220, Jun. 2008.

[4] C. A. Micchelli, Y. Xu, and H. Zhang, "Universal kernels," *The Journal of Machine Learning Research*, vol. 7, p. 26512667, 2006.

[5] X. Xu, I. W. Tsang, and D. Xu, "Soft margin multiple kernel learning," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 24, no. 5, p. 749761, May 2013.

[6] R. Poli, W. Langdon, N. McPhee, and J. Koza, *A field guide to genetic programming*, 2008.

[7] T. Howley and M. G. Madden, "The Genetic Kernel Support Vector Machine: Description and Evaluation," *Artificial Intelligence Review*, vol. 24, no. 3-4, pp. 379–395, Nov. 2005. [Online]. Available: http://www.springerlink.com/index/10.1007/s10462-005-9009-3

[8] K. M. Sullivan and S. Luke, "Evolving kernels for support vector machine classification," in *Proceedings of the 9th annual conference on Genetic and evolutionary computation - GECCO '07*. ACM Press, 2007, p. 1702.

[9] L. Dioan, A. Rogozan, and J.-P. Pecuchet, "Improving classification performance of Support Vector Machine by genetically optimising kernel shape and hyper-parameters," *Applied Intelligence*, vol. 36, no. 2, pp. 280–294, Oct. 2010.

[10] P. Koch, B. Bischl, O. Flasch, T. Bartz-Beielstein, C. Weihs, and W. Konen, "Tuning and evolution of support vector kernels," *Evolutionary Intelligence*, vol. 5, no. 3, pp. 153–170, May 2012.

[11] T. Evgeniou, M. Pontil, and T. Poggio, "Regularization networks and support vector machines," *Advances in Computational Mathematics*, vol. 13, pp. 1–50, 2000.

[12] A. Ekrt and S. Z. Nmeth, "Maintaining the diversity of genetic programs," in *Genetic Programming*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2002, vol. 2278, pp. 162–171.

[13] R. A. Horn and C. R. Johnson, *Matrix analysis*. Cambridge university press, 2012.

[14] E. K. Burke, S. Gustafson, and G. Kendall, "Diversity in genetic programming: An analysis of measures and correlation with fitness," *IEEE Transactions on Evolutionary Computation*, vol. 8, no. 1, pp. 47–62, 2004.