A Learning Approach for Ill-Posed Optimisation Problems

Jörg Frochte[1] and Stephen Marsland[2]

[1] Bochum University of Applied Sciences, D 42579 Heiligenhaus, Germany
joerg.frochte@hs-bochum.de
[2] Victoria University of Wellington, NZ stephen.marsland@vuw.ac.nz

Abstract. Supervised learning can be thought of as finding a mapping between spaces of input and output vectors. In the case that the function to be learned is multi-valued (so that there are several correct output values for a given input) the problem becomes ill-posed, and many standard methods fail to find good solutions. However, optimisation problems based on multi-valued functions are relatively common. They include reverse robot kinematics, and the research field of AutoML – which is becoming increasingly popular – where one seeks to establish optimal hyperparameters for a learning algorithm for a particular problem based on loss function values for trained networks, or to reuse training from previous networks. We present an analysis of this problem, together with an approach based on k-nearest neighbours, which we demonstrate on a set of simple examples, including two application areas of interest.

Keywords: Multi-valued functions · Ill-posed optimisation · Local models · AutoML

1 Introduction

Consider a standard regression problem. Given a training set $S = (x, y)$ of pairs of input and target data, we aim to identify a function $h(x)$ such that

$$y = h(x) + \varepsilon, \tag{1}$$

where the ε term gives the residual error and is a function of the data and the particular form of regression used.

Implicit in this model is the assumption that the problem is well-posed, i.e.,

1. a solution exists,
2. the solution is unique, and
3. the solution is stable (its behaviour changes continuously with respect to the initial conditions).

Ill-posed problems – a class that includes many inverse problems – can be very important. We will focus on problems that do not have a unique solution, so that there is more than one possible target for a given input, which are

known as multi-valued functions. As shall be discussed shortly, this class of problems includes inverse kinematics and a form of automated machine learning (AutoML).

Standard regression techniques do particularly poorly on these problems, finding solutions between the two possible correct results. For example, consider the cylindrical spiral given by $y^2 = \tan^{-1}\left(\frac{x_2}{x_1}\right), \quad x_1, x_2 \in [-1, 1]$, which is a variant on that suggested in [9]. Note that this corresponds to the mapping between Cartesian and polar coordinates. As can be seen on the left of Figure 1, there are two correct targets for each input pair (x_1, x_2), and the multilayer Perceptron (MLP) fails to find either solution. Local approaches such as k-nearest neighours also fail, albeit in a slightly different way. We will revisit this problem in Section 3.1, but on the right of Figure 1 demonstrate that our approach finds both solutions to this multi-valued problem.

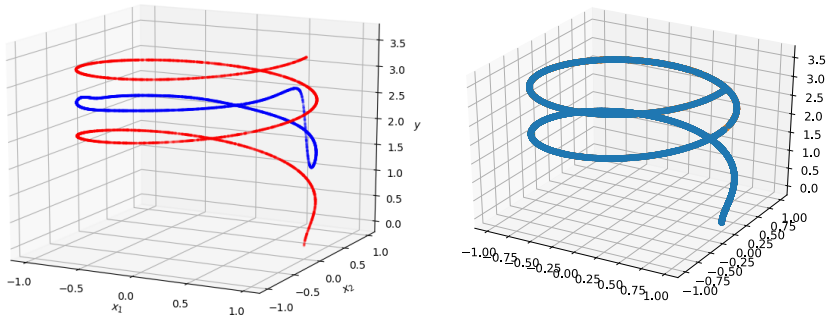


Fig. 1. *Left:* The MLP (blue) finds an incorrect solution to a multi-valued function consisting of the spiral shown in red. *Right:* Our approach finds both solutions.

The heart of the problem lies in the fact that so-called multi-valued functions define a left-total binary relationship [8]. A binary relationship takes ordered pairs $(x \in X, y \in Y)$ for sets X and Y, and assigns it to some subset of the Cartesian product $X \times Y = \{(x, y) : x \in X, y \in Y\}$. As it is defined on ordered pairs, there can be many y that match an x and vice-versa, and there may also be some $y \in Y$ that do no have a relationship with any x and vice versa; see also [5], which considers the learning of such relationships. A left-total binary relationship requires that at least one y exists for each x, but does not require that it is unique. Obviously, such functions are not globally invertible.

This is why inverse problems so commonly have this form: consider the inverse kinematics problem from robotics. In this classic regression problem, the old adage that all roads lead to Rome (i.e., there are many paths to reach any given location of the end-effector) means that the inverse is not properly defined. While adding regularisation functions such as aiming to minimise energy or time can

help, there is still no guarantee of a unique, and therefore potentially invertible, solution.

One way to deal with this lack of global invertibility is to appeal to the implicit function theorem, and create local invertible approximations of the function. Providing that the regions of the domain with matching points in the codomain are sufficiently separated, this can work well, as was shown in [6] where networks of radial basis functions were used to approximate the multi-valued function. However, there will always by pathological examples where this assumption breaks down. As was discussed in [6], there is no guarantee that an appropriate partition of the data can be found, and even if there is, the implicit function theorem requires that the function is continuously differentiable, which may not be true in general.

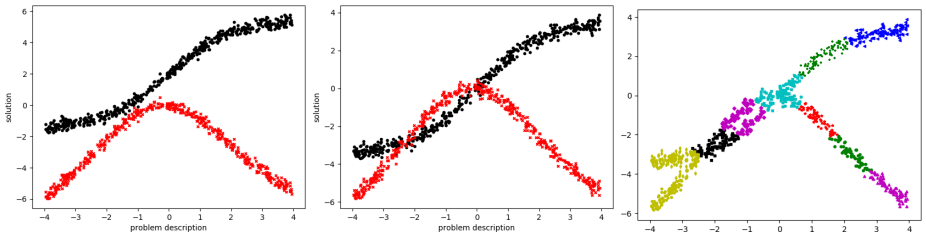


Fig. 2. Trying to separate solutions by clustering is not easy if the relationship is complicated, as in the centre picture. DBSCAN produces the set of clusters shown on the right.

In fact, this problem can be made worse if the data is clustered first to aid in the selection of partitions, as Figure 2 shows. If both solutions (red and black) are easy to separate, as in the left problem set, clustering works well. If their relationship is more complicated, as in the middle of the figure, one ends up with a lot of problems. The right plot shows the result of using DBSCAN [4], which is dependant on the parameters. As one can see, the subsets identified by clustering do not necessarily consist of data from one solution (which we call 'pure'). Increasing the number of clusters makes them pure, but tends towards local learners.

For some problems there is additional structure to the problem that can be exploited. This is true for inverse kinematics, where there is a time progression that gives order to the data samples: the data is generated as a sequence $f_i : t \mapsto \mathbb{R}^n = y$ of several functions f_i, each providing a mapping between the same start and end points. The set of different trajectories performing the same mapping provide enough structure for neural networks to find good approximations, such as the RNNs used by [13] and the standard feedforward networks of [9] and [1]. Beyond this, [11] presents an approach using regularisation networks that includes learning an algebraic representation of the multi-valued function, while in [2] the authors extend a hierarchical Dirichlet process hidden Markov model

to a multi-valued function regression. One of the most recent publications in this branch is [3], which uses an approach based on an infinite mixture of linear experts, thus enabling online learning.

Consider now the example of automatic machine learning (AutoML), which aims to automate the whole process of selecting machine learning algorithms, fitting hyperparameters, and optimising them, see e.g., [14]. One way to formulate this problem is as an optimisation problem. Data are presented as triples consisting of a description of a learner such as a neural network, a set of weight values for it, and the value of the loss function of that network on some dataset: $(\theta, x, f_D(x, \theta))$, where the D subscript labels the dataset that was used for testing. A set of such triples comprise the training set for an optimisation problem. In more general terms, this can be written as $\min_x f(x, \theta)$, where $f(\cdot)$ is the objective function (e.g., the neural network loss function, or some function to minimise such as energy, or cost), x are the variables we wish to optimise over (the weights of the neural network), and θ is a set of parameters that specify the precise problem. Note that this formulation includes multi-objective optimisation automatically using some of the elements of vector θ as weights for these parts of the objetive function, e.g., $f(x, \theta) = \theta_1 f_1(x, \theta) + \theta_2 f_2(x, \theta)$.

In general, the dataset consists of noisy samples, such as the weights and loss function values of trained neural networks with particular sets of hidden layers, or certain examples of inverse kinematic solutions. In neither case is it likely that the dataset contains the actual global optimum.

We now present an extension to the k-nearest neighbour algorithm for optimisation of multi-valued functions, and demonstrate that it is well-behaved with respect to hyperparameter choices on a variety of test cases, including a simple example from AutoML.

2 A clustering-based algorithm for multi-valued functions

Our algorithm is a variant of the partition approach that was described previously. For a given value of vector θ, we identify the points in the dataset that are closest (in the Euclidean norm) to that value and then cluster those points into c or fewer clusters based on a weighted sum of their coordinates using kNN (where $d_i = \theta - \theta_i$ and smear is a parameter that smoothes the kNN regression):

$$x(\theta) = \sum_{i=1}^{k} \omega_i x_i \quad \text{with} \quad \omega_i = \frac{(d_i + \frac{\text{smear}}{k})^{-1}}{d} \quad \text{and} \quad d = \sum_{i=1}^{k} \left(d_i + \frac{\text{smear}}{k} \right)^{-1} \tag{2}$$

The pseudocode for our algorithm is given in Algorithm 1. The algorithm works in three stages: we find a large set of points close to θ, and then refine it to the points in that set with smallest $f(\theta, x)$ values. These points are then clustered into c clusters, which are post-processed to remove clusters with fewer than k members, and retain the k points that are nearest with respect to θ for the rest. It is not necessary to specify c exactly, as is shown in Figure 3.

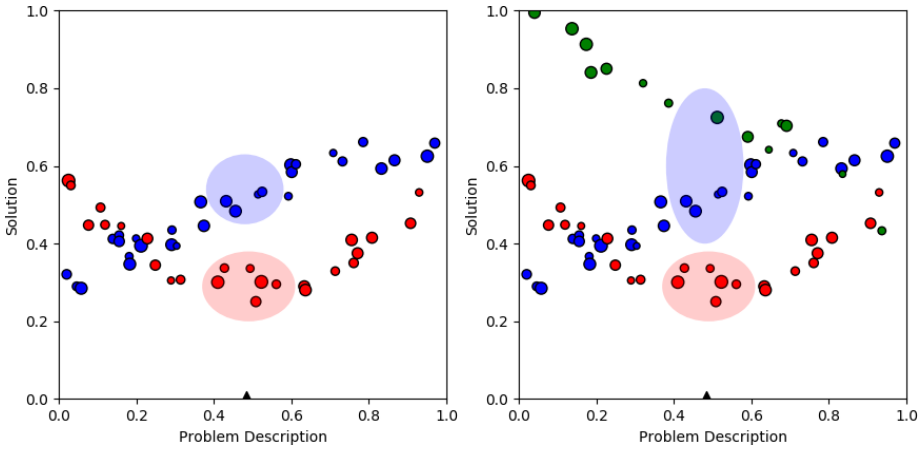


Fig. 3. It is not necessary to identify the number of clusters precisely. The algorithm is searching for points close to the θ value marked with a black triangle. *Left:* When there are two clusters, setting $c = 2$ means that two clusters are identified. However, *right:* the same parameter value when there are three clusters will merge two of the clusters, resulting in a spread of potential points. However, when the points nearest to each other in every cluster are identified, only the blue dots in the blue cluster will be considered.

The k samples in a cluster are used with (2) to find the regression value of $f(x, \theta)$. The values at $f(x_i, \theta)$ could be used as weights, but we found that this did not improve the results much, and was computationally more expensive.

In the clustering step it is possible to use any clustering algorithm. We have tested k-means and fuzzy C-means, as they have a small number of hyperparameters and distribute points across as many clusters as possible. Note that although kNN is often described as a lazy learner, it is common to build a kd-tree or similar of all the distances to enable nearest neighbours to be found as efficiently as possibly. This is the approach used in e.g., scikit-learn [10].

Finally, we note that although the algorithm is intended to work with the function values $f(x, \theta)$ (score), it is possible to use it in cases where this information is not available. In that case $n = 1$, and there is no need to perform line 3 and the regression of f at the final step in line 7. The first example in the next section considers this type of example.

3 Experiments and Analysis

3.1 Regression without score

We first consider the example shown in the Introduction, in order to show that the algorithm is equivalent to other multi-valued function learners if the function values $f(x, \theta)$ are not provided. We sampled 20,000 points from the curve shown

Algorithm 1 The kNN-MV Algorithm

procedure kNN-MV($x, \theta, f(x, \theta)$)
Require: $k \geq 1$, $c \geq 1$ ▷ k = #neighbours, c = #output classes
Require: $2 \leq m \leq 4$, $1 \leq n \leq 3$ ▷ m and n control #points in neighbourhoods
  $\hat{N}$ = the indices of the $a \cdot k \cdot m \cdot n$ nearest neighbours of θ
  N = the $a \cdot k \cdot m$ examples in $\hat{N}$ with lowest values of $f(x, \theta)$
  cluster the points in N into c clusters according to distance with respect to x ▷ e.g., using k-means
  for each cluster **do**
    if cluster has $< k$ members **then** return $\emptyset$
    else use the k points closest to each other for the regression of θ and $f(x, \theta)$ using (2)
    end if
  end for
end procedure

in red in Figure 1, adding noise at the boundaries 0 and $\sqrt{4\pi}$, as they tend to overlap, leading to more choices, and set $c = 2$.

Table 1 shows that the algorithm works well and is stable with respect to added noise; see also the right of Figure 1. It is hard to compare exactly with [9], as not all details are provided there, but they report a root mean square error (RMSE) of about $1.85 \cdot 10^{-2}$, which is comparable to ours, and we make fewer assumptions about the structure of the solution.

Table 1. Results using the parameters $k = 3$, smear= 1, $a = 2$, $m = 4$.

	Percentage of noise on x in training data				
	0%	1%	2%	3%	4%
one correct answer	100 %	100 %	100 %	100%	100%
two correct answers	91.2 %	92.8%	91.0%	91.9 %	89.9%
AME	$6.5 \cdot 10^{-4}$	$8.2 \cdot 10^{-3}$	$1.6 \cdot 10^{-2}$	$2.5 \cdot 10^{-2}$	$3.2 \cdot 10^{-2}$
RMSE	$8.1 \cdot 10^{-4}$	$1.1 \cdot 10^{-2}$	$2.2 \cdot 10^{-2}$	$3.4 \cdot 10^{-2}$	$4.4 \cdot 10^{-2}$

3.2 An analytical test case

As a simple example of the core usage of our method, we considered the function:

$$\begin{aligned} f(x, \theta) = 3 &- \exp(20(-(x_1 - \theta_1)^2 - (x_2 - \theta_2)^2)) \\ &- \exp(20(-(x_1 - \theta_2 + 1)^2 - (x_2 - 0.5\theta_1^2)^2)) \\ &- 0.95 \exp(20(-(x_1^2 - \theta_2) - (x_2 + \theta_3 + 0.5)^2)) \end{aligned}$$

The problem consists of finding the multiple solutions to $\arg\min_{x \in \mathbb{R}^2} f(x, \theta)$ for fixed $\theta \in \mathbb{R}^3$.

We created 429,618 samples by computing numerical solutions by gradient descent from random initial starting points. Note that the space also has local minima, and so many of these numerical solutions will have become stuck in them. Table 2 shows the results with $k = 3$ and $k = 5$ for both the standard kNN and our multi-value version, while Figure 4 looks at the effect of the two parameters, m and n. The table shows that the method is far more successful that standard kNN, even at finding one solution, while the figure demonstrates that the algorithm is not very sensitive to the choice of the parameters. In general, $1.5 \leq n \leq 2.5$ and $2 \leq m \leq 4$ is a good range. Beyond this one can see in table 2 that using the score in kNN-MV, which means in this application the value of the function f, leads to better results.

Table 2. Results using kNN and kNN-MV ($k = 3$ $k = 5$, $m = n = 2.5$).

Method	**% answers with**		**average error**
	one result	**both results**	**on answers**
Standard kNN (k=3)	21.70%	00.00%	0.0042
kNN4MV (k=3) without score	89.44%	62.12%	0.0072
kNN4MV (k=3) with score	99.82%	83.20%	0.0083
Standard kNN (k=5)	13.11%	00.00%	0.0357
kNN4MV (k=5) without score	91.85%	64.48%	0.0073
kNN4MV (k=5) with score	99.92%	85.35%	0.0094

3.3 Shot on Goal Learning

We now present a more interesting example, which combines kinematics – as common application area – with an ill-posed optimisation problem. Suppose that a robot is aiming to kick a ball into a soccer goal in such a way that it goes over the goalkeeper's head, as shown on the left of Figure 5, by learning from examples. We model the path of the ball via a non-linear ordinary differential equation that includes the four fixed components of θ shown on the right of the figure with their allowable ranges. In fact, we will allow the wind speed θ_w to vary, and assume that the agent knows only the sign of it, not the value. The controllable parameter is the velocity of the ball $x(t)$ at $t = 0$. We represented $x(0)$ in polar coordinates as (φ, s) pairs of direction and speed.

We judged that a goal was scored if the ball arrives at the the goal at a height between 2.1 and 2.4 metres above the ground, with angle of travel $\varphi < -0.5$ degrees. We created a large number of examples that satisfied these criteria, and gave them a score based on their height and angle when the ball crossed the goalline: $f(x, \theta) = x_2 + |\varphi|/2$, i.e., height + half the angle.

There are two very different strategies that can achieve this successfully, corresponding to two different angles of initial velocity. The higher angle strategy receives higher scores, but because the ball is in the air longer, it is more affected by the random wind. We computed 30,000 successful goal scoring examples by

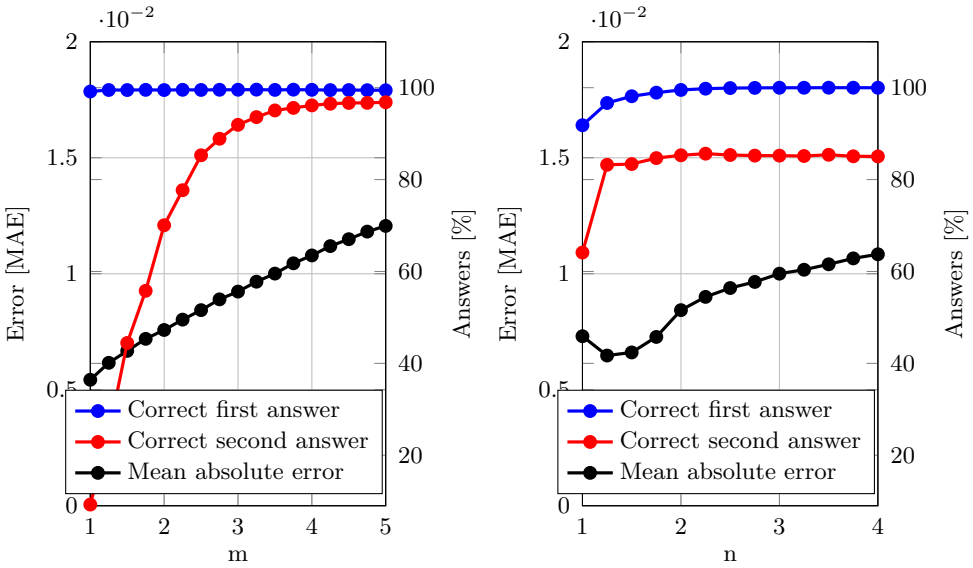
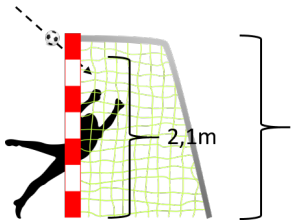


Fig. 4. Results of varying m with a constant $n = 2$ (*left*) and varying n for a constant $m = 2.5$ (*right*).



$\theta_w \in [-4, 4]$[m/s] (wind speed)
$\theta_m \in [0.25, 0.75]$ [kg] (ball mass)
$\theta_r \in [0.05, 0.2]$ [m] (ball radius)
$\theta_d \in [10, 80]$ [m] distance to goal

Fig. 5. Schematic showing a successful shot on goal, and the parameters of θ with their allowable range.

brute force as an initial training set. Our approach to this was as follows: (φ, s) pairs and values for θ were chosen by kNN-MV and evaluated. Those that were successful in scoring a goal were saved, with their score. For the others, we performed a search around that φ of $\pm 5°$, reducing that to $\pm 1°$ as more entries were added to the dataset, and finally stopping using the search at all.

The top line of Figure 6 shows the final distribution of samples in the training set for two parameters of θ, ball radius and distance to the goal, while the bottom line shows them with respect to the two components of x, φ and s. Samples were created uniformly at random, but only successful samples were added to the database. Hence the non-uniform distributions with respect to these variables suggest where there are fewer successful solutions, and hence the problem is harder.

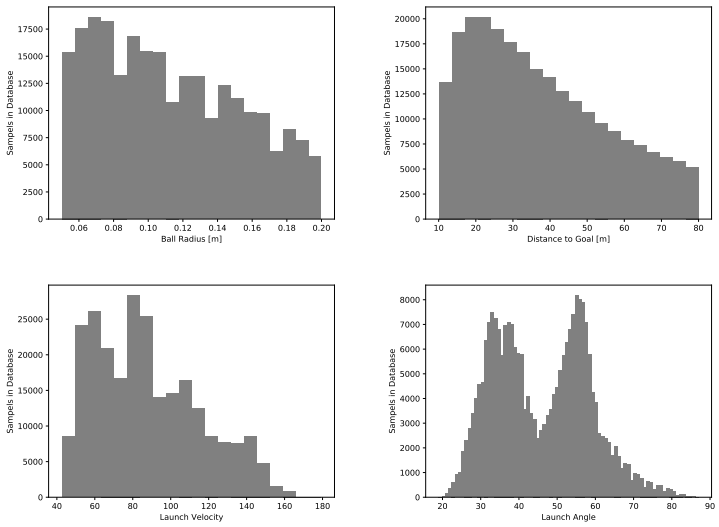


Fig. 6. Histogram of the distribution of solutions in the training database with respect to *top row:* fixed parameters (for a datapoint) θ_r and θ_d and *bottom row:* solution parameters φ and s.

We train a multi-layer Perceptron, kNN, and kNN-MV (both with $k = 5$) using the dataset created above. To test the algorithms, we ran each of them as follows. A random configuration was chosen for θ. Three of the values were then held fixed, and four samples of wind strength θ_w were chosen (which the learner does not know), all with the same sign (which the learner does know). The learner than generated four choices of (φ, s). The percentage of successful shots, and the score achieved by the most successful shot, are given in Table 3. Note that this is not an easy problem, and we consider that the near 40% of kNN-MV is a good result.

Table 3. Results of learning agent using three supervised methods for different limits for the distance to the goal.

Method	**Maximum Distance**									
	20m		30m		40m		60m		80m	
	%	score	%	score	%	score	%	score	%	score
MLP	4.74	2.71	0.21	2.70	0.60	2.70	0.67	2.68	1.65	2.68
kNN	15.11	2.70	11.83	2.68	9.81	2.66	8.35	2.65	6.88	2.64
kNN-MV	36.82	2.60	39.14	2.59	37.12	2.59	33.20	2.59	27.30	2.58

3.4 A Simple AutoML Example

The fact that a multi-layer Perceptron with a single hidden layer of 2 nodes and a total of 9 weights can solve the XOR problem is well-known. There are actually six different solutions for the weights, up to scaling, which arises because there are three different ways to construct XOR from more basic logical operators:

$$x_1 \,\mathrm{XOR}\, x_2 = (x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge x_2) \tag{3}$$
$$= (x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2) \tag{4}$$
$$= (x_1 \vee x_2) \wedge \neg(x_1 \wedge x_2) \tag{5}$$

The first bracket term is represented by one hidden unit and the second by another, while the final AND/OR-operation is performed by the output layer. Hence, the network has three ways to choose the weights, and a symmetry in the order of the operators.

One goal of AutoML is to reuse old models to speed-up the design of new ones. Learning the weights of a neural network involves solving a multi-valued problem because different configurations of the weights can result in very similar performances, see e.g. [7], [12].

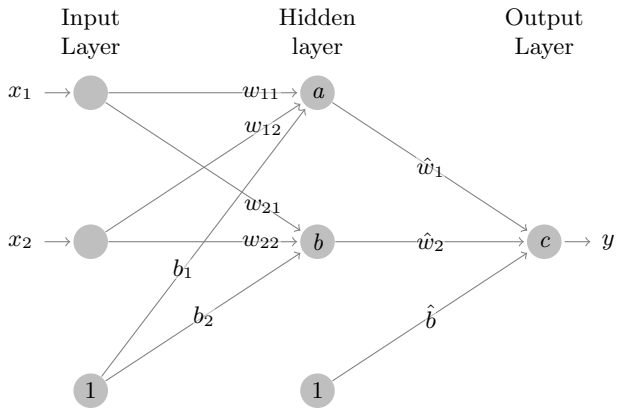


Fig. 7. MLP with one Hidden Layer for XOR

We took a standard XOR problem, and applied a rotation $\alpha \in [-45°, +45°]$ and a translation $[t_1, t_2]$, with $0 \leq t_i \leq 0.5$. The result is the set illustrated in figure 8.

Our AutoML problem is to find values for the 9 weights of the neural network (see Figure 7) based on a set of trained networks for different values of $\theta = (\alpha, t_1, t_2)$ with the assistance of values $f(x, \theta)$ being the value of the loss function (sum-of-squares error). The database only contains solutions with $f(x, \theta) < 0.1$.

Table 4 shows the results on this example problem for training databases with 500 and 1000 samples in the training set. In each case kNN-MV was able

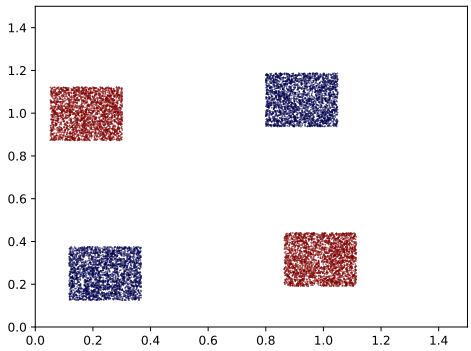


Fig. 8. Affine transformation of an XOR Training Set.

to find at least one solution. The algorithm produced 600 solutions during the test, since there are 6 possible different weights (up to scaling). *% sol.* is the percentage of those solutions that were found. As mentioned before, kNN-MV can predict the quality of the solutions as well. Therefore the column $|q - \text{loss}|$ is the mean difference between the predicted value of the loss function and the real one during the test, where loss is the mean value of the loss function for that solution. Finally, *class* is 1 if all are classified correctly and 0 if none are.

Table 4. Success of KNN-MV ($k = 5$) of different sizes of training databases, all values are mean average over 100 test problems.

samples in training set							
500				1000			
% sol.	$\lvert q - \text{loss}\rvert$	class	loss	% sol.	$\lvert q - \text{loss}\rvert$	class	loss
89.6%	0.11	0.94	0.170	90.3%	0.0479	0.98	0.0998

4 Conclusions

In this paper we have considered the setting of multi-valued functions in general, and then presented a kNN variant that learns about these functions in general, and for ill-posed optimisation application cases in particular. Our algorithm shows good results on a set of example problems, which includes a simple application in AutoML. We have shown that a global approach decomposing the multi-value functions cannot be performed without making strong assumptions concerning the nature of the database, e.g. time-series data. Nevertheless, one of our future prospects is to use the presented method as the nucleus of a mixed lazy and eager learner with the goal of achieving higher-order regression on trusted subsets of the database. We will also be applying our algorithm to

real-world, higher-dimensional datasets. The used datasets and simulation codes in this paper are published on the authors website.

References

1. Brouwer, R.K.: Feed-forward neural network for one-to-many mappings using fuzzy sets. Neurocomputing **57**, 345–360 (2004)
2. Butterfield, J., Osentoski, S., Jay, G., Jenkins, O.C.: Learning from demonstration using a multi-valued function regressor for time-series data. In: Humanoid Robots (Humanoids), 2010 10th IEEE-RAS International Conference on. pp. 328–333. IEEE (2010)
3. Damas, B., Santos-Victor, J.: Online learning of single-and multivalued functions with an infinite mixture of linear experts. Neural computation **25**(11), 3044–3091 (2013)
4. Ester, M., Kriegel, H.P., Sander, J., Xu, X.: A density-based algorithm for discovering clusters in large spatial databases with noise. In: Proceedings of the Second International Conference on Knowledge Discovery and Data Mining. pp. 226–231 (1996)
5. Goldman, S.A., Rivest, R.L., Schapire, R.E.: Learning binary relations and total orders. SIAM Journal on Computing **22**(5), 1006–1034 (1993)
6. Hahn, K., Waschulzik, T.: On the use of local rbf networks to approximate multivalued functions and relations. In: L. Niklasson, M.B., Ziemke, T. (eds.) Proceedings of the 8th International Conference on Artificial Neural Networks (ICANN). pp. 505 – 510. Springer (Sep 1998)
7. Hecht-Nielsen, R.: Theory of the backpropagation neural network. In: Neural networks for perception, pp. 65–93. Elsevier (1992)
8. Kilp, M., Knauer, U., Mikhalev, A.: Monoids, Acts and Categories, With Applications to Wreath Products and Graphs. de Gruyter (2000)
9. Lee, K.W., Lee, T.: Design of neural networks for multi-value regression. In: Neural Networks, 2001. Proceedings. IJCNN'01. International Joint Conference on. vol. 1, pp. 93–98. IEEE (2001)
10. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: Machine learning in Python. Journal of Machine Learning Research **12**, 2825–2830 (2011)
11. Shizawa, M.: Multivalued regularization network-a theory of multilayer networks for learning many-to-h mappings. Electronics and Communications in Japan (Part III: Fundamental Electronic Science) **79**(9), 98–113 (1996)
12. Sussmann, H.J.: Uniqueness of the weights for minimal feedforward nets with a given input-output map. Neural networks **5**(4), 589–593 (1992)
13. Tomikawa, Y., Nakayama, K.: Approximating many valued mappings using a recurrent neural network. In: Neural Networks Proceedings, 1998. IEEE World Congress on Computational Intelligence. The 1998 IEEE International Joint Conference on. vol. 2, pp. 1494–1497. IEEE (1998)
14. Wong, C., Houlsby, N., Lu, Y., Gesmundo, A.: Transfer learning with neural automl. In: Advances in Neural Information Processing Systems. pp. 8356–8365 (2018)