Function and Pattern Extrapolation with Product-Unit Networks

Babette Dellen[1(✉)], Uwe Jaekel[1], and Marcell Wolnitza[1,2]

[1] Department of Mathematics and Technology, RheinAhrCampus Remagen, University of Applied Sciences Koblenz, Joseph-Rovan-Allee 2, 53424 Remagen, Germany
[2] Third Institute of Physics - Biophysics, Georg-August-University Göttingen, Friedrich-Hund-Platz 1, 37077 Göttingen, Germany
{dellen,jaekel,wolnitza}@hs-koblenz.de

Abstract. Neural networks are a popular method for function approximation and data classification and have recently drawn much attention because of the success of deep-learning strategies. Artificial neural networks are built from elementary units that generate a piecewise, often almost linear approximation of the function or pattern. To improve the extrapolation of nonlinear functions and patterns beyond the training domain, we propose to augment the fundamental algebraic structure of neural networks by a product unit that computes the product of its inputs raised to the power of their weights. Linearly combining their outputs in a weighted sum allows representing most nonlinear functions known in calculus, including roots, fractions and approximations of power series. We train the network using stochastic gradient descent. The enhanced extrapolation capabilities of the network are demonstrated by comparing the results for a function and pattern extrapolation task with those obtained using the nonlinear support vector machine (SVM) and a standard neural network (standard NN). Convergence behavior of stochastic gradient descent is discussed and the feasibility of the approach is demonstrated in a real-world application in image segmentation.

Keywords: Product units · Neural network · Function extrapolation

1 Introduction

Complex machine learning problems that have been intractable in the past are now being solved using deep neural networks performing fast computations on parallel hardware [9–11,15,20]. Standard neural networks eventually perform a piecewise, almost linear approximation of the function or pattern that is to be learned [8,13,14,16,18,19,21]. This limits extrapolation of nonlinear relationships into areas of the feature space that are not covered by the training data. However, in many applications, extrapolating and making predictions from data is exactly what is required. In the past, solutions of this problem have been sought by nonlinearily transforming the data implicitly or explicitly into a higher

J. M. F. Rodrigues et al. (Eds.): ICCS 2019, LNCS 11537, pp. 174–188, 2019.
https://doi.org/10.1007/978-3-030-22741-8_13

dimensional space [5,17] or by adding nonlinear units to the network representing terms of higher-order polynomials [4]. The disadvantage is that either the basis of this nonlinear space or a nonlinear kernel has to be provided beforehand. Alternatively, it would have to be adapted to the problem at hand by considering an exponentially increasing repertoire of combinatorial combinations of them, leading quickly to a computationally intractable situation.

In this paper, we follow a different approach: We expand the algebraic capabilities of the neural network by integrating input multiplication at the operational level through product units [6,12], representing a computational multiplicatory analogon to the classical summation unit, the McCulloch Pitts neuron (McP) [13] (see Fig. 1a–b). We combine the output of several product units by means of a classical summation unit into a generalized algebraic operations network (GAON) (see Fig. 1c), which then can be used to build larger networks (see Fig. 1d).

Product units have been first introduced in the past to learn the higher-order input combinations of nonlinear problems such as the parity problem [6,12]. However, for the rather simple networks investigated in these studies, difficulties were arising when using backpropagation to train those networks. For example, it was reported that the parity-8 problem could not be trained using backpropagation [12]. In that particular case, the neural network consisted of a single product unit. It was assumed that the solution space for product units is too convoluted, giving rise to many local minima. Weight initialization turned out to be especially difficult for product-unit networks. This discouraging result probably explains why the idea of product-unit neural networks was not taken much further in the following years.

Our work differs from those approaches in several ways: (i) We only use positive inputs to avoid difficulties otherwise arising for exponents representing roots. This is achieved by taking the absolute value of the input before feeding it into the product unit. For problems containing negative values, we first feed the input to a standard summation layer (which is trained together with the nonlinear network) to transform the input data into the positive domain. (ii) When solving classification problems, the exponents (or weights) of the product unit can only take positive values. This avoids exploding terms for small input data, but does not limit the applicability of the approach (see Discussion) (iii) We consider only networks consisting of substructures of several product units, i.e., the GAON (see Fig. 1c). This property allows a larger range of functions to be described by the network. (iv) For the parity-8 problem and a real-world labeling problem with mutliple classes we choose a network in which the output of several GAONs is fed into a standard summation layer with softmax activation. Decision hypersurfaces emerge as intersections of GAON functions in the data space. Different from standard neural networks, those hypersurfaces can be highly nonlinear. (v) We explore especifically the extrapolation capabilities of product-unit networks and compare them to the ones of standard neural networks [8,13,14,16,21] and the nonlinear support-vector machine [5,17].

For the problems and the network architectures that we studied for this work, we did not encounter problems with convergence. This is not in contradiction with the studies described before [6,12]. Our network is larger and employs the product units in a different way.

2 Methods

In this work, we propose to augment the algebraic structure of neural networks by allowing generalized multiplicatory operations to take place at the level of their basic computional units. Artificial neural networks consist of simple computational units that receive inputs from many other units of the same kind (see Fig. 1a) [8,13,18] and form a weighted sum of these inputs that is passed through a threshold or other rectifying function to still other units of the same kind, which process their inputs in like manner. This elementary unit is known as the McCulloch Pitts neuron (McP) [13]. In the standard neural network, these units are arranged in layers, and information is flowing only in a single direction through the layers [19]. Learning is usually performed by adjusting the weights through error backpropagation after definition of a suitable loss (or cost) function [21]. Let $\mathbf{x} = (x_1, \ldots, x_n)$ be an n-dimensional input vector that is supplied to an McP unit; then the net output of this unit (prior to any rectification step) is defined by $o_{\text{sum}}(\mathbf{x}) = \sum_{i=1}^{n} w_i x_i$, where the w_i, $i = 1, \ldots, n$ are the weights of the connections (see Fig. 1a). Solutions can be thought of as a division of the data space by a hyperplane [18], with class regions corresponding to the resulting half spaces. If one seeks to separate regions that are defined by highly nonlinear boundaries, many hyperplanes are needed to carve the hypervolume corresponding to the class-defining region. This can be achieved by arranging many McP neurons in a layer and combining their outputs again using an McP unit, representing the output unit of the resulting *multilayer perceptron* [19].

In contrast to the classical approach, we aim at separating the class-defining regions by nonlinear hypersurfaces. To achieve this, we use product units [6, 12], similar to the McP unit, that perform algebraic multiplication instead of summation (see Fig. 1b, left panel), yielding the output $o_{\text{mult}}(\mathbf{x}) = \prod_{i=1}^{n} x_i^{w_i}$, where the w_i, $i = 1, \ldots, n$ are real numbers, representing the weights of the connections.

From a computational point of view, the operation performed by the multiplication unit can be implemented through an equivalent network consisting of a balanced arrangement of McP units with logarithmic and exponential activation functions (see Fig. 1b, right panel). Switching from summation to multiplication can be achieved by taking the logarithmic values of the inputs, i.e., $\log x_1, \ldots, \log x_n$, and applying the exponential function to the net output, yielding

$$\begin{aligned} o_{\text{mult}}(\mathbf{x}) &= \exp\left(\sum_{i=1}^{n} w_i \log x_i\right) \\ &= \prod_{i=1}^{n} x_i^{w_i}. \end{aligned}$$

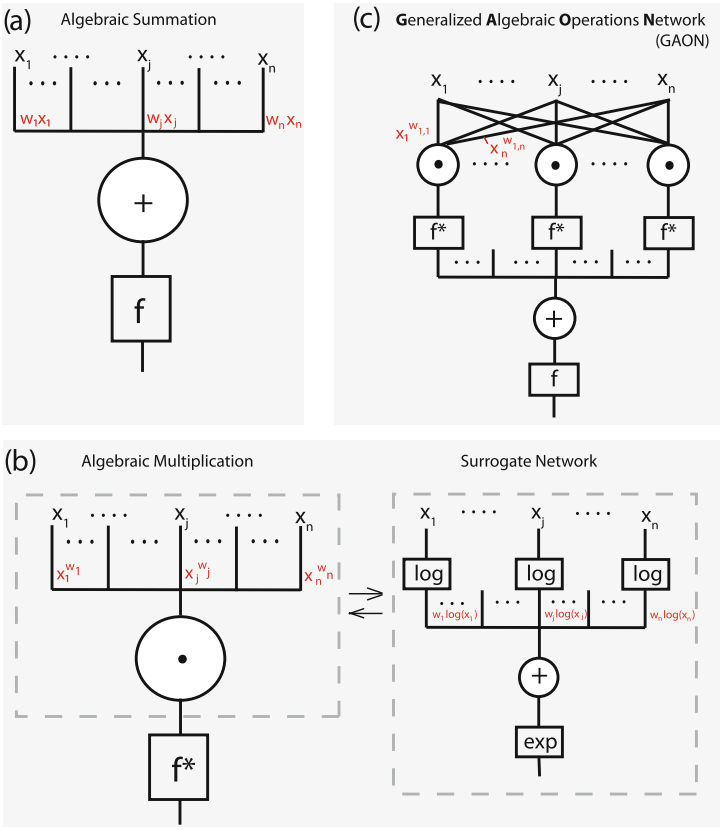

Fig. 1. (a) Algebraic summation by the classical McCulloch-Pitts (McP) unit. (b) Left panel: Algebraic multiplication unit: Inputs are raised to the power of their weights and then multiplied, followed by an activation f^* [6,12]. Right panel: Algebraic multiplication (gray broken line) can be implemented by a surrogate network with alternating log and exp activation functions. Here, the absolute values of the inputs are taken and a small shift is added to avoid zero values in the input. (c) Generalized algebraic operations network (GAON) built from elementary summation and multiplication units. (d) Larger network including a hidden layer of GAONs. The first dense layer allows transforming the input data into a suitable domain. Several parallel GAONs implement nonlinear input-output relations and their output is passed on a dense summation layer performing softmax activiation for classification purposes.

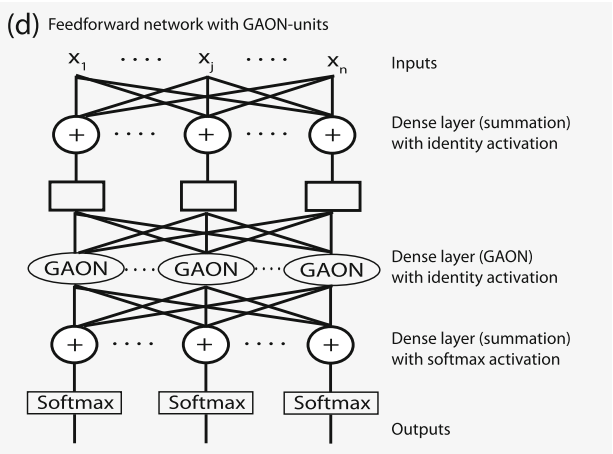

Fig. 1. (*continued*)

To approximate nonlinear functions of the type

$$s(x_1, ..., x_n) = \sum_{k=1}^{m} w_k \left[\prod_{i=1}^{n} x_i^{w_{k,i}} \right],$$

we arrange m or more multiplication units in a layer (the hidden layer) and connect it to an McP neuron, i.e., a summation unit (see Fig. 1c). This arrangement defines the generalized algebraic operations network (GAON). We choose the identity as activation function f^* for the hidden multiplication units of the GAON. Then, the net output of the hidden layer of the GAON is given by

$$o_{\mathrm{GAON}}(\mathbf{x}) = \sum_{k=1}^{m} w_k \left[\prod_{i=1}^{n} x_i^{w_{k,i}} \right],$$

where the $w_{k,i}$ are weights from the input units to the hidden units of the GAON, and the w_k are the weights from the hidden units to the output node of the GAON. The final output of the GAON is then $\hat{y} := f\left(o_{\mathrm{GAON}}(\mathbf{x})\right)$, where f is the activation function of the output unit. For function approximation tasks, we choose f to be the identity. For classification tasks, we choose the activation function to be $f(b) = 1/(1 + e^{-b})$. This form of network not only allows representation of any polynomial of n-input variables, but also fractions and roots. Functions that can be described by power series (such as cosine and sine) could potentially be approximated by the network if a large enough number of hidden multiplication units is used. Training is performed using error backpropagation [19,21]. The loss functions and further details are provided in the appendix. Furthermore, we integrate GAON-units into larger networks (see Fig. 1d). First, the input is fed into a dense layer of summation units with identity activation. A bias input is included. Then, the output of the dense layer is processed by

a dense layer of parallel GAON-units. The GAON-layer output provides input to another dense layer of summation units with softmax activation. We use this network to solve multi-label classification tasks.

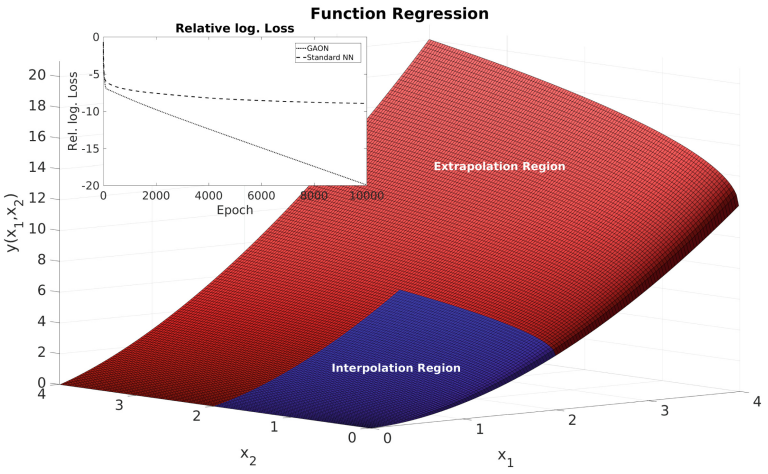

Fig. 2. Surface plot of the function $y = x_1\sqrt{x_2}+x_1^{1.8}$. Blue colors indicate the data used for training of the methods, the red colors represent the test data used for evaluation. Inset: The relative logarithmic values of the loss function for the regression task during training are plotted for the standard NN and the GAON. (Color figure online)

3 Results

3.1 Function Regression

We first test the network of Fig. 1c having $m = 2$ hidden units on a function-regression task. For this purpose, we generate a data set consisting of feature vectors (x_1, x_2) and output values $y = x_1\sqrt{x_2} + x_1^{1.8}$. This function is plotted in Fig. 2. The input values of the training data are in the range of $x_1, x_2 \in [0, 2]$, while those of the test data cover the larger ranges $x_1, x_2 \in [0, 4]$. The respective areas of the function are plotted in blue (training) and red (testing) color. In this way we can evaluate the extrapolation capacity of the method. Using stochastic gradient descent, the algorithm converges to a stable solution after less than 100 epochs, as seen in the inset of Fig. 2, where the relative logarithmic loss is plotted as a function of the training epochs. We observed similar convergence behavior for other choices of the input-output relationship. For comparison, we use a standard neural network with one hidden layer of 10 hidden McP units (standard NN). The standard NN is trained in the same way as the GAON on the same training set (see appendix) [19]. As seen in Fig. 2, the value of the loss function of the standard NN remains larger than that of the GAON even after many epoch. We should note however that the learning rate of the GAON is

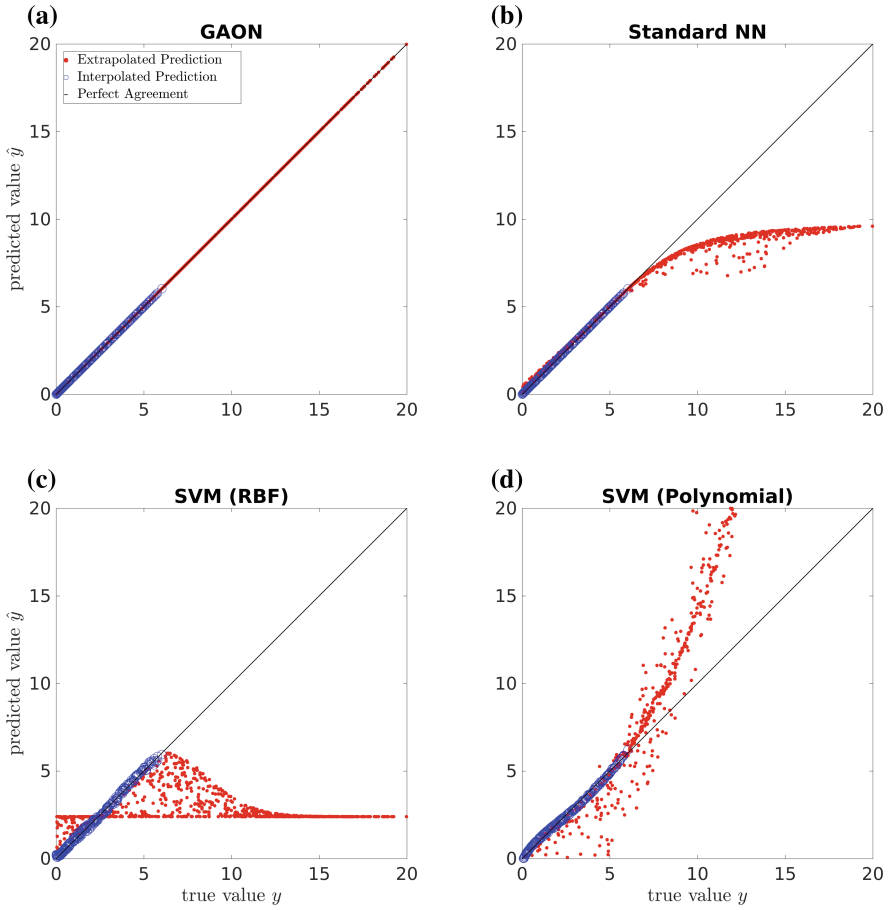

Fig. 3. Function-regression task. The function $y(x_1, x_2) = x_1\sqrt{x_2} + x_1^{1.8}$ is learned and the predicted value is plotted against the true value for (a) the GAON (b), the standard NN, (c) the SVM with RBF kernel, and (d) polynomial kernel. (Color figure online)

slower than that of the standard NN (by a factor of ≈ 0.1), because the GAON is more sensitive to weight changes than the standard NN (see appendix). This results in a larger number of iterations per epoch for the GAON.

We further compare the extrapolation capabilities of our method with those of the standard NN and the nonlinear support-vector machine (SVM), the latter for both polynomial and radial-basis-function kernels (RBF) [3,19]. The results are shown in Fig. 3. To evaluate the results, we plot the predicted output $\hat{y}$ as a function of the true value y for the different methods. In case of perfect agreement, the points should lie on a line, i.e., $\hat{y} = y$ (shown in black). The results are plotted as blue circles for interpolated values (i.e., values that lie within the range of the training data) and as red dots for extrapolated values

(situated outside that range). It can be seen that the results from the GAON lie close to the line of perfect agreement for both interpolated and extrapolated predictions (Fig. 3a). For the standard NN (Fig. 3b) and the SVMs (Fig. 3c-d), the extrapolated predictions mostly disagree with the observed values.

3.2 Classification

We further test the network on a data-classification task. Here, we use $m = 10$ hidden units for the GAON, but similar results are obtained for other choices of m (not shown). In this task, vectors belonging to class +1 (TRUE) have end points lying within an area of symmetrical shape, namely a circle, a rectangle, and a diamond, defined by the implicit function $|x_1|^p + |x_2|^p \leq r^p$ with $p = 0.5, 2, 20$, respectively. The vectors lying outside of the shape belong to class 0 (FALSE). Since the shapes are symmetric, it suffices to train and test the methods on the absolute values of the input data. To specifically test their extrapolation capabilities, we train the methods only on partial shapes. The respective region of feature space used for training is underlaid in gray color in the plots (see Fig. 4). We repeat the training and testing process 100 times, varying randomly drawn training sets and computing the frequency with which the data point is assigned to the class TRUE. The frequency of occurrence is color-coded in shades of blue and presented together with the correct class boundaries of the classification problem (black line). In Fig. 4a, the results from the GAON are shown for the different shapes. In areas where the patterns have to be extrapolated, our method outperforms the standard NN (Fig. 4b) and the two SVMs (Fig. 4c–d) for all cases.

Table 1. Classification results for the different methods and data sets. The number of support vectors is averaged over 100 trials.

Method	Accuracy [%]			
	Total	Interpolation	Extrapolation	# Hidden Units/Support Vectors
p = 0.5				
GAON	97.68 ± 0.12	99.64 ± 0.01	91.82 ± 0.48	10
Standard NN	97.66 ± 0.04	99.65 ± 0.01	91.69 ± 0.16	10
SVM (RBF)	94.65 ± 0.04	99.04 ± 0.02	81.49 ± 0.14	352.97 ± 7.23
SVM (Polynomial)	94.40 ± 0.37	99.43 ± 0.02	79.30 ± 1.46	8.49 ± 0.08
p = 2				
GAON	98.80 ± 0.09	99.72 ± 0.01	96.04 ± 0.35	10
Standard NN	97.80 ± 0.11	99.72 ± 0.01	92.03 ± 0.44	10
SVM (RBF)	97.30 ± 0.03	99.50 ± 0.01	90.69 ± 0.13	239.13 ± 4.90
SVM (Polynomial)	98.22 ± 0.24	99.61 ± 0.02	94.03 ± 0.93	6.63 ± 0.09
p = 20				
GAON	99.22 ± 0.02	99.29 ± 0.01	99.00 ± 0.06	10
Standard NN	98.46 ± 0.11	99.74 ± 0.01	94.63 ± 0.45	10
SVM (RBF)	97.76 ± 0.03	99.40 ± 0.02	92.83 ± 0.12	258.01 ± 5.64
SVM (Polynomial)	98.85 ± 0.05	99.44 ± 0.02	97.07 ± 0.16	9.42 ± 0.16

The accuracy of the classifications results obtained with the different methods for the three shapes is presented in Table 1. The GAON achieves higher accuracies than the standard NN and the two SVMs in the extrapolation task for $p = 0.5$, $p = 2$ and $p = 20$. The GAON outperforms all three methods in the extrapolation task. In the interpolation task, the standard NN and the GAON achieve comparable accuracies.

3.3 Real-World Application

As an illustrative example for a real-world application of product-unit networks, we considered the problem of completing incomplete labelings of images as they occur frequently in image-segmentation tasks. In Fig. 5a the image of a tobacco plant during plant growth is shown [2]. Using graph-based image segmentation [7] and removing segments below a critical size threshold, an imcomplete labeling of the image is obtained (see Fig. 5b). Areas without label are depicted in black color. The number of labels corresponds to the number of prominent leaves in the images plus the background. The leaf pointing north is particularly poorly segmented by the method. Completing the labeling of the image can be posed as a classification task, where the pixel coordinates and their respective class labels ranging from 0 to 5 provide the input training data for the classifier. Pixels, for which no label could be assigned, are not a part of the training data.

We use these data to train the product-unit network shown in Fig. 1d. The two-dimensional input data, i.e., the pixel coordinates, provide the input to a dense layer consisting of two summing units without activation. A bias is provided to each of the summing units to allow shifts of the data. The output of the two summing units is fed to a layer of six GAON-units. The output of the GAON-layer then is passed on to another layer of six summing units with softmax activation. For the loss function, Tensorflow's sparse categorical crossentropy is used [1]. Using gradient descent with batches of 40 data points, we obtained an accuracy of 0.9979 after 2000 epochs. We use the neural network to extrapolate class labels into the previously unlabeled regions and obtain the completed segmentation shown in Fig. 5c. The segments now assume the nonlinear, approximately elliptic shape of the leaves.

4 Discussion

In this work, we augmented the algebraic structure of neural networks with a multiplicatory elementary unit [6,12] that computes the products of its inputs raised to the power of their weights, representing the multiplicatory analogon to the classical McP-neuron. To evaluate the extrapolation capabilities of the network, we used both a function-extrapolation task and a pattern-classification-extrapolation task. We could demonstrate that the GAON outperforms the standard NN and the nonlinear support vector machine for the given tasks. We presume that the gain in performance originates from the property that the

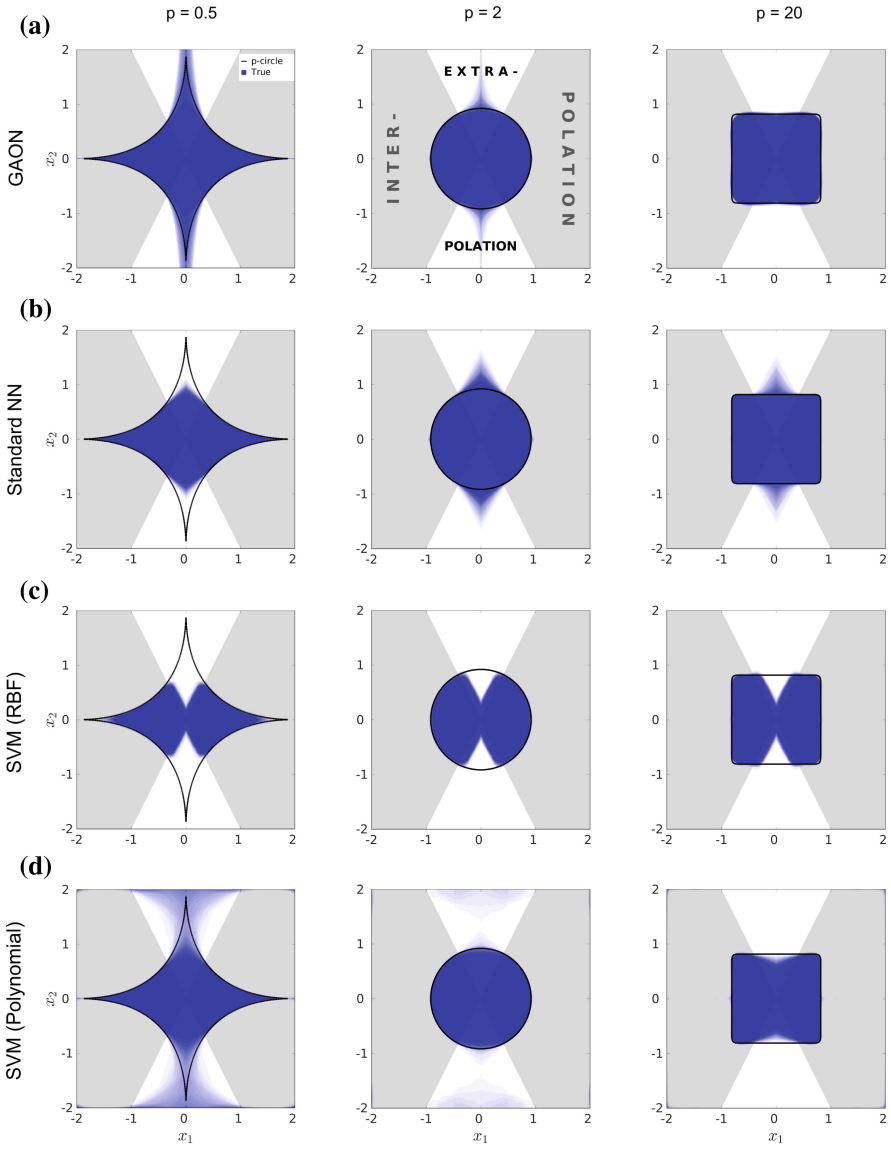

Fig. 4. Classification results for (a) the GAON, (b) the standard NN, (c) the SVM with RBF kernel, and (d) polynomial kernel for the different data sets. The frequency of occurrence of the class label TRUE is color coded in shades of blue for 100 trials and shown together with the correct class boundaries (black line).

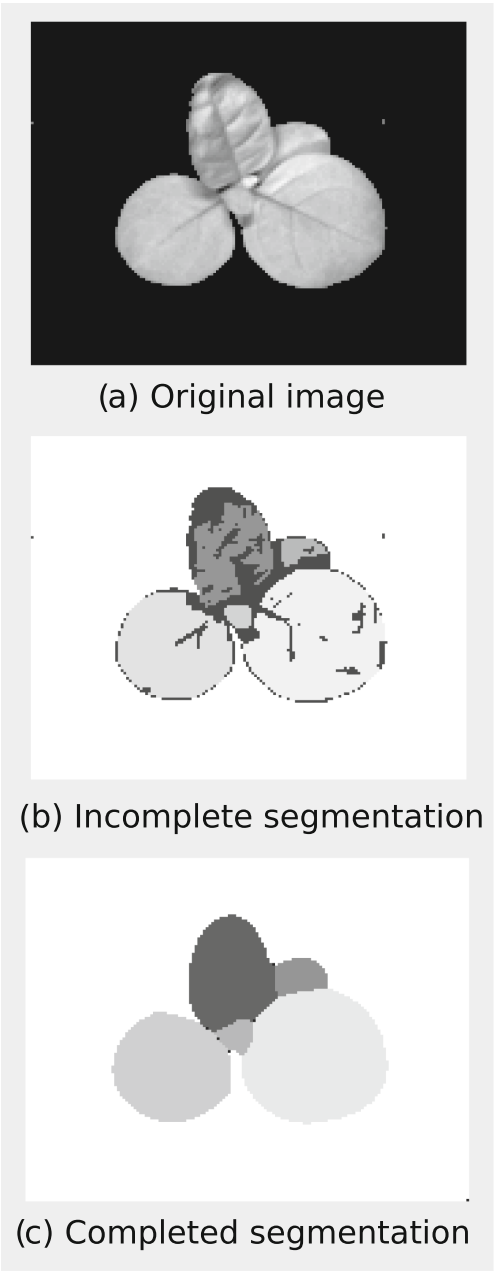

Fig. 5. (a) Grayscale image of a tobacco plant [2]. (b) Incomplete segmentation obtained with a graph-based image-segmentation method [7]. (c) Completed segmentation obtained with a product-unit network trained with the incomplete-segmentation result (for better visualization of the results, the color code of the segment labels has been slightly modified).

multiplier network can learn arbitrary functions of the form

$$\sum_{k=1}^{m} w_k \prod_{i=1}^{n} x_i^{w_{k,i}},$$

being generated by the outputs of the hidden units $\prod_{i=1}^{n} x_i^{w_{k,i}}$, allowing extrapolation into unknown domains. The net outputs of the hidden units of the standard NN are linear functions and as such do not represent a generating set, hindering extrapolation of the pattern. The nonlinear support vectors machine is constrained by the choice of the kernel function; the polynomial kernel SVM performs excellently for the circular shape in our example, but in this case the kernel represents an almost perfect fit for the characteristics of the pattern.

Our proposed network is structurally simple and the training straightforward. The use of adjustable weights that are appearing as exponents of the inputs in the network allows the GAON to learn arbitrary functions from the data. This property makes it fundamentally different from polynomial classifiers, where a fixed set of predefined polynomial basis functions is chosen for the discrimination function and combined in a weighted sum [19]. Related to this, Bayesian neural networks employing higher-order polynomials [4] have been proposed in the past, for which however similar restrictions apply.

We restricted ourselves to positive weights in the data classification task. This prevents failure when input values are near zero. However, this does not limit the applicability of the method by any means. We explain this giving an example: The inequality $\frac{1}{x_1} + x_2^2 \leq 1$ contains a negative exponent, but it can be recast by multiplying both sides with x_1 (given that we work with positive input data only, which can always be achieved by shifting). This yields the new inequality $1 + x_1 x_2^2 \leq x_1$, or $x_1 x_2^2 - x_1 \leq -1$, which still represents the same classification problem as the original inequality. Hence, there exists an infinite set of discrimination functions with positive exponents that describe the classification problem. More than that, we suspect that this property allows the network to escape local minima by moving toward a more advantageous discriminant function, securing convergence of the method. This conjecture might further explain why the multiplier network is robust to changes in the number of hidden units when solving classification tasks. However, when solving regression tasks, extrapolation performance depends on the number of hidden units, and we observed that extrapolation improves when the number of hidden units matches the number of basis functions of the function space required for properly describing the input-output relationship. In all cases our method leads to a very sparse representation of functions and patterns that allows, unlike nonlinear SVMs, a direct interpretation of its components in terms of mulitplicative relationships.

We further demonstrated that product units can be intergrated in larger networks (see Fig. 1d) using the framework Tensorflow [1] by extrapolating labels in image segmentation into unlabeled regions (see Fig. 5). This is particular useful when the shapes of the objects that are to be segmented are unknown prior to the task. Our approach can be applied to other labeling problems as well, e.g., the assignment of disparity labels to pixels in stereo vision.

To address concerns regarding convergence of gradient-descent training, we revisited the parity-8 problems [12] and trained the same network that was used to solve the image-segmenation task (see Fig. 1d) for the parity-8 problem. The method converged to solutions with accuracy 1. The authors of [12] searched for a single set of weights that would classify the parity-8 data correctly by considering a network that basically consisted of a single product unit only. In a larger product-unit network, there presumably exists more than one weight combination that classifies the data correcty. We reckon that this has an impact on convergence.

Acknowledgements. We thank John W. Clark for his help in improving the manuscript through his comments. B.D. and U.J. contributed in equal parts to the ideation, design and implementation of the method. M.W. contributed to the implementation, integration and the training/testing of the method.

A Appendix

All methods are implemented in MATLAB. We use a training set size of 1000, drawn uniformly from randomly distributed values in the range $0, \ldots, +2$ for the regression task and $-2, \ldots, +2$ for the classification task. Every input is mapped onto the first quadrant by using only absolute values. To test the inter- and extrapolation capabilities of our network, we require the examples of the training set to satisfy the following condition: $|x_1| > 0.5|x_2|$. This defines our interpolation region (represented as the white space in Fig. 4). The examples of the test set are allowed to take on any value in the range indicated above. We use self-coded functions for the GAON and standard NN and built-in functions from Matlab for the SVMs.

We also implemented the GAON in Python within the Keras-API of the Tensorflow library [1] to enable fast computation with graphics processing units (GPUs) and make use of its modularity. We used this framework to build the larger product-unit network used for multi-label classification.

A.1 GAON

For training in function-regression tasks, we use $\sum_{j=1}^{N}(\hat{y_j} - y_j)^2$ as loss function, where y_j is the true value corresponding to the input vector $\mathbf{x}_j$ and $\hat{y_j}$ the predicted value. For network training in classifications tasks, we use the cross entropy $\epsilon = -\sum_{j=1}^{N}[y_j \log \hat{y_j} + (1 - y_j) \log (1 - \hat{y_j})]$ as loss function. N is the number of training vectors.

For our runs with the GAON we use a constant learning rate of $l_r = 10^{-3}$ and train the network for 10000 epochs with stochastic gradient descent, using one training example per iteration. One epoch is defined as a complete process over the whole training set (corresponding to 1000 iterations in our setup). The initial values for the weights in the first layer are drawn randomly from a

uniform distribution, whereas those of the second layer are normally distributed. Additionally we restrict the weights of the first layer to be only positive in the training process (see discussion).

For the results of the classifications (Fig. 4 and Table 1) we repeat the whole training and testing process 100 times with varying training sets and present the mean along with the standard error of the mean.

A.2 Standard Neural Network

For our runs with the standard neural network we use a constant learning rate of $l_r = 10^{-2}$. We train the network with stochastic gradient descent for 10000 epochs with one example per iteration. The initial values for the weights are drawn in the same as in the GAON method, but here we do not restrict the weights in any layer. We used the same loss functions as for the GAON.

A.3 Support Vector Machine

For our runs with the SVMs we use the Matlab-function *fitrsvm* for the regression and *fitcsvm* for the classification task. Our models include standardization (centering and devision by standard deviation) of the input and an automatic selection of the scaling factor by a heuristic procedure. The polynomial SVM has an order of four.

References

1. Abadi, M., et al.: TensorFlow: Large-scale machine learning on heterogeneous systems (2015). https://www.tensorflow.org/
2. Agostini, A., Alenyà, G., Fischbach, A., Scharr, H., Wörgötter, F., Torras, C.: A cognitive architecture for automatic gardening. Comput. Electron. Agric. **138**, 69–79 (2017)
3. Bose, B., Guyon, I., Vapnik, V.: A training algorithm for optimal margin classifiers. In: Proceedings of the 5th Annual Workshop on Computational Learning Theory, pp. 144–152 (1992)
4. Clark, J.W., Gernoth, K.A., Dittmar, S., Ristig, M.L.: Higher-order probabilistic perceptrons as bayesian inference engines. Phys. Rev. E **59**, 6161–6174 (1999)
5. Cortes, C., Vapnik, V.: Support vector network. Mach. Learn. **20**, 273–297 (1995)
6. Durbin, R., Rumelhart, D.E.: Product units: a computationally powerful and biologically plausible extension to backpropagation networks. Neural Comput. **1**(1), 133–142 (1989)
7. Felzenszwalb, P.F., Huttenlocher, D.P.: Efficient graph-based image segmentation. Int. J. Comput. Vis. **59**(2), 167–181 (2004)
8. Hubel, D., Wiesel, T.: Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. J. Physiol. Lond. **160**, 54–106 (1962)
9. Husain, F., Dellen, B., Torras, C.: Scene Understanding Using Deep Learning. Academic Press, San Francisco (2017)
10. Husain, F., Schulz, H., Dellen, B., Torras, C., Behnke, S.: Combining semantic and geometric features for object class segmentation of indoor scenes. IEEE Robot. Autom. Lett. **2**, 49–55 (2017)

11. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. Nature **521**, 436–444 (2015)
12. Leerink, L.R., Giles, C.L., Horne, B.G., Jabri, M.A.: Learning with product units. Adv. Neural Inf. Process. Syst. **7**, 537 (1995)
13. McCulloch, W.S., Pitts, W.: A logical calculus of the ideas immanent in nervous activity. Bull. Math. Biophys. **5**, 115–133 (1943)
14. Minsky, M.L., Papert, S.: Perceptrons : An Introduction to Computational Geometry. MIT press, Cambridge (1988)
15. Rajaraman, S.K., Antani, S., Candemir, S., Xue, Z., Kohli, M., Thoma, G.: Comparing deep learning models for population screening using chest radiography. SPIE Med. Imaging **10575E**, 1–11 (2018)
16. Rosenblatt, F.: The perceptron - a probabilistic model for information storage and organization in the brain. Psychol. Rev. **65**, 386 (1958)
17. Scholkopf, B., Smola, A.J.: Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond. MIT Press, Cambridge (2001)
18. Theodoridis, S., Koutroumbas, K.: Linear Classifiers, 4th edn. Academic Press, Boston (2009)
19. Theodoridis, S., Koutroumbas, K.: Nonlinear Classifiers, 4th edn. Academic Press, Boston (2009)
20. Silver, D., et al.: Mastering the game of go with deep neural networks and tree search. Nature **529**, 484–489 (2016)
21. Werbos, P.J.: Beyond regression : new tools for prediction and analysis in the behavioral sciences. Harward University (1974)