





# Approximate Function Classification

Martin Lukac<sup>1</sup>, Krzysztof Podlaski<sup>2</sup>, and Michitaka Kameyama<sup>3</sup>

<sup>1</sup> Nazarbayev University, Nur-sultan, Kazakhstan  
martin.lukac@nu.edu.kz

<sup>2</sup> University of Lodz, Lodz, Poland  
krzysztof.podlaski@uni.lodz.pl

<sup>3</sup> Emeritus Professor Tohoku University, Sendai, Japan  
kameyama@ecei.tohoku.ac.jp

**Abstract.** Classification of Boolean functions requires specific software or circuits to determine the class of a function or even to distinguish between two different classes. In order to provide a less costly solution, we study the approximation of the NPN function classification by a artificial neural network (ANN), and shown that there are configurations of ANN that can perfectly classify four-bit Boolean functions. Additionally, we look at the possibility of learning the classification of four-bit Boolean functions using a set of three-bit Boolean neural classifiers, and determine the scalability. Finally we also learn a discriminator that can distinguish between two functions and determine their similarity or difference in their NPN classes. As a result we show that the approximate neural function classification is a convenient approach to implement an efficient classifier and class discriminator directly from the data.

**Keywords:** Function classification · Approximate learning · Neural networks

## 1 Introduction

The classification of functions has several applications in both industry and research. Knowing the properties of groups of functions allows one to study solutions to a problem from a group-like perspective. This includes the cost of circuit estimation or group functions implementations. It also helps the industry to design circuit from an approximate cost: functions grouped into specific groups can have similar realization properties and thus similar and predictable cost [1, 10, 12]. Finally, function classification can be used to discover structures in the groups of functions that are not completely understood [2, 4, 5].

The classification can be performed by a standard look-up or by a functional approach. However, such a direct approach is not well scalable because it requires the definition of all possible input value combinations. Therefore alternative methods for generating classes of functions using approximate methods are desired. The classification of functions using machine learning has been previously explored [6–9, 11]. However, most of the available works focus on learning Boolean functions using machine learning.

The function classification becomes function approximation when the classification is a many to one mapping. In particular, when the classification results in group properties such that the cost of any function within the group is the same, then a logic design process can be simplified by searching for equivalence groups rather than for single functions. The most recognized classifications of Boolean functions are P-equivalent, NP-equivalent, NPN-equivalent [5].

Two Boolean functions are NPN-*equivalent* if they can be transformed into each other by one or more of the following transformations: the negation of any input variable, permutation of input variables, and negation of the function.

In the paper, we consider the possibility of classification of functions using off-the-shelf machine learning approaches [6]. The target of such classification will show if the NPN classification can be efficiently learned. Therefore, if the groups' properties that represent linear transformations can also be easily represented in the embedded feature space. In addition, we are also interested to see if we can learn the classification of NPN classes on a subset of functions and then generalize it to other functions not used for learning. The expected result will show if such an approximate learning and NPN classes representation can be used to learn classifications of functions where there are too many samples to enumerate. Finally, we consider the proposed method as a general template for classification: the NPN classification presented in this paper is only a starting point for a more general classification framework.

The machine learning approach to classification requires learning data. For functions with more than four input bits, the number of NPN classes and the number of functions is too large to be used for learning. Therefore, we look at the problem from the point of view of learning  $n$ -bit NPN classes by composing the machine learning classifier from  $n - 1$ -bit functions. We train accurate  $n - 1$  NPN function classifiers, and by combining several of them, we train the  $n$  bit classifier on partial data. We show under which conditions such an approach is possible and what amount of data is required for accurate prediction. We also investigate the discrimination of NPN classes by a trainable neural network. We show that a single network is quite capable to distinguish between functions of different classes without intermediary determination of their respective NPN classes. Therefore, using the neural methods for distinguishing NPN classes provides a considerable advantage when compared to classical, circuit-based methods.

## 2 Experiments

### 2.1 Datasets

For the experiments, we have created two datasets. The first dataset is the set of all four-bit irreversible functions, and the mapping to be learned is  $\mathcal{M} : \{0, 1\}^{2^4} \rightarrow \mathbb{L}_4$ , with  $\mathbb{L}_4$  being the set of all NPN classes for four-variable irreversible functions. There are 222 NPN classes for four-variable irreversible functions, and the number of data samples in the dataset is 65536.

The second set of datasets contains NPN labels and functions for one, two, three, and four input bits. The target of this dataset is to use it to learn and

to determine the scalability hardness of the mapping from a set of functions of  $n - 1$ -variables to  $n$ -NPN labels:  $\mathcal{N} : \{0, 1\}^{n-1} \times \dots \times \{0, 1\}^{n-1} \rightarrow \mathbb{L}_n$ . The  $\mathbb{L}_n$  represents the labels of  $n$ -bit functions.

## 2.2 Four-Variable Irreversible Logic Functions

In preliminary experiments, we evaluated various machine learning algorithms: multi-layer perceptron (MLP), support vector machine (SVM), decision tree (DT), and random forests (RF), but only the MLP showed good enough learning convergence and final classification accuracy. Therefore, we decided to analyze if it's possible to obtain a perfect classifier using MLP and what architecture is required. For the experiment, we took 4-bit Boolean functions. The training set contains 65% of all functions in the dataset, and a test set contains the rest. We discovered that by learning the default network for up to 300 epochs, we could not obtain 100% positive answers on the test set with one fully connected hidden layer MLP with up to 120 neurons. Therefore we have added a second hidden layer and performed a grid search.

**Table 1.** Accuracy of MLP classifier with two hidden layers,  $l_1, l_2$  denote sizes of first and second hidden layer respectively.

$l_1 \setminus l_2$	10	15	20	25	30	35	40	45	$l_1 \setminus l_2$	50	55	60	70	75	80	85	90
10	0.128	0.171	0.250	0.234	0.450	0.295	0.267	0.240	80	0.997	0.995	<b>1.000</b>	0.999	0.999	<b>1.000</b>	0.997	0.998
30	0.173	0.436	0.718	0.791	0.804	0.823	0.829	0.829	90	0.992	0.991	<b>1.000</b>	0.996	<b>1.000</b>	<b>1.000</b>	0.998	<b>1.000</b>
40	0.159	0.700	0.767	0.814	0.809	0.821	0.895	0.865	100	0.998	0.996	0.996	0.999	0.999	0.999	0.999	<b>1.000</b>
60	0.341	0.547	0.787	0.922	0.889	0.963	0.968	0.950	110	0.998	0.994	0.999	0.999	<b>1.000</b>	0.997	<b>1.000</b>	0.996
70	0.211	0.444	0.760	0.919	0.971	0.945	0.987	0.995	120	0.996	0.999	0.996	<b>1.000</b>	0.999	<b>1.000</b>	<b>1.000</b>	<b>1.000</b>

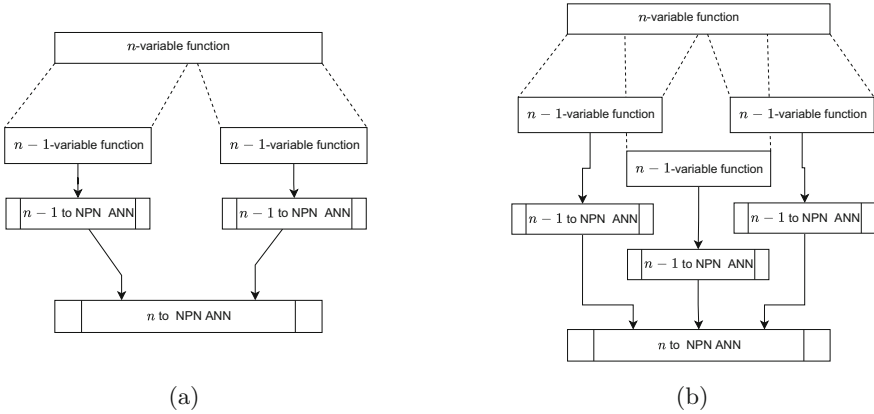
The parameters of the network and the obtained accuracy on the training set after learning 300 epochs are presented in Table 1. The rows of Table 1 show the number of neurons in the first hidden layer, and the columns show the number of neurons in the second hidden layer. As can be seen, the full precision of the NPN classification was achieved when the first layer had at least 80 neurons and the second one at least 60. It agrees with the observation that the MLP is able to approximate the multivariate function only with two hidden layers [3].

## 2.3 From $n - 1$ to $n$ Classification

Classifying functions with more than four bits is more difficult due to very large number of functions and therefore does not allow efficient learning. Consequently, we evaluate alternative models to classify NPN labels for functions of  $n$  bits as a function of various compositions of functions of  $n - 1$  bits.

The main idea of this approach is shown in Fig. 1: a pre-trained neural network for a  $n - 1$  NPN classification is used as a module (component neural network) for predicting  $n - 1$  NPN labels from a  $n$ -variable subfunction extracted from the target  $n$ -variable function. Each component network has as inputs the

bits representing a  $n - 1$ -variable function and outputs an NPN class. The results of these component networks are then used to train a  $n$  bit NPN classifier. The main concept is to extract a set of  $n - 1$ -variable NPN function labels from the  $n$ -variable function and then use these labels to predict the  $n$ -variable NPN label. Note that in Fig. 1, each component network is the same, and only the last layer (denoted  $n$  to NPN) of the whole network is trained on the  $n$ -variable function dataset.



**Fig. 1.** Schematic diagram of estimating the NPN classes of a four-variable functions using pretrained classifiers for the NPN classes of a three bit logic function.

There are in total  $\binom{2^n}{2^{n-1}}$  possible selection of inputs to each component network. However, we are mainly interested in selecting  $2^{n-1}$  out of  $2^n$  values in a continuous manner. Therefore we tested three very simple configurations that contain: two, three, and five-component neural networks.

The input to the network is represented by  $2^n$  values and the  $2^{n-1}$  values for the components networks were extracted using positional indexes. For the two network models, each component network accepts  $2^{n-1}$  bits at positions  $0$  to  $2^n/2 - 1$  and  $2^n/2$  to  $2^n - 1$  respectively. For three-component networks, the first two-component networks take bits in a similar fashion to the two-component network model. The third-component network uses bits starting from the  $2^n/4$  and ending at  $2^n/4 + 2^n/2$  bit. For the five-component networks model, the fourth and fifth-component neural networks are simply added to the three component neural network model. The fourth and fifths components ANN had inputs fed from the  $2^n/8$  to  $2^n/8 + 2^n/2$  bit and from the  $2^n/2 - 2^n/8$  to  $2^n - 2^n/8$  bit respectively. As an example, Fig. 1 shows the three component model classifier: the blocks labeled  $n - 1$ -bit function represents the functions of  $n - 1$ -bits extracted from  $n$ -variable input function and the blocks labeled  $n - 1$  to NPN represents the pre-trained NPN component network classifiers for  $n - 1$ -variable functions.

The component networks were trained independently and prior to the final experiment. In the presented case the component networks were trained on three-variable Boolean functions and their NPN labels. In this case the mapping is  $\mathbb{K} : \{0, 1\}^3 \rightarrow \mathcal{J}_3$ . The three-variable NPN classification networks were trained up to 100% accuracy. Each component network had two hidden layers with 50 and 20 neurons, respectively, and the  $n$ -bit NPN classifier network had the same structure as well.

**Table 2.** Results of learning NPN classification for four-variable Boolean functions using three-bit neural NPN classifiers.

Model	3 bit NPN ANN specs	Accuracy	4 bit NPN ANN specs	Accuracy
2 subnets	50/20	1	50/20	0.679 @ 0.7
3 subnets	50/20	1	50/20	0.769 @ 0.65
5 subnets	50/20	1	50/20	0.87 @ 0.26

The results of these experiments are shown in Table 2. Each row shows the result of a particular configuration and its experimental accuracy. The second and fourth columns show the model’s configuration: column two shows the component network configuration, while column four shows the whole model configuration. Column three shows the component network accuracy; it is always one because the networks were pre-trained to the highest accuracy. Column five shows the accuracy of the whole model. The results are reported as accuracy@train-set size. The first observation is that the accuracy increases when more component networks are used. It is natural as there are only thirteen NPN labels for functions of three bits. Therefore, using two components networks does not generate enough output combinations to reach the required 222 NPN classes of four-variable functions. Therefore the most accurate result is obtained with five components networks. The second observation is that less data is required when more component networks are provided. As can be seen with two-component networks the accuracy of 67% was reached at  $r_e = \frac{0.3}{0.7}$  but the highest recorded accuracy was 87% for five-component networks at  $r_e = \frac{0.74}{0.26}$ . Therefore the observation seems to lead to a conclusion that knowledge from functions on less bits can be efficiently used to learn the classification of larger functions with a fraction of training data.

### 3 NPN Classes Discrimination

The final set of experiments is aimed at finding out at what cost a machine learning approach can be used to distinguish two functions from different or the same NPN class. The most natural approach is to classify two functions directly using a binary classifier. The inputs to such a model are two  $2^n$  long

binary vectors representing two  $n$ -variable functions. The problem that we are thus facing is a solution to a mapping  $\mathcal{J} : \{0, 1\}^{2^n} \times \{0, 1\}^{2^n} \rightarrow \{0, 1\}$ .

Remembering that the number of functions grows rapidly with the number of bits, already for four-variable functions, the number of samples is  $(2^{16})^2$ . We implemented a pseudo-random method, allowing us to quickly get a large number of samples. Not all combinations can be generated. Therefore, four different datasets were generated: each dataset being a multiple of the total number of functions of  $n$ -variables. For each dataset a same size test dataset was generated so that the train and evaluation minimizes the amount of overlap between the training and testing data. The experiment was run with a multi-layer perceptron with 33 neurons in the input layer, 100 neurons in the hidden layer and two neurons in the output layer. Each experiment was run for 2000 learning epochs.

The first experiment was performed only with the training dataset. Thus, the learning and the evaluation were performed using the same dataset. The reason for this was to determine the learning difficulty. The summary of this first learning experiment for comparing four-variable NPN functions is shown in Table 3. As can be seen, the accuracy of the discrimination remains constant with the increasing size of the dataset. Each dataset contains more instances of similar and dissimilar pairs of functions. Interestingly the learning occurred at only a very high learning rate  $\lambda = 0.1$ . Such high  $\lambda$  is unusual considering that standard experiments in machine learning use much lower values for the learning rate. Also, it is worth noting that the learning accuracy grows with the increasing epochs. With the current model, the accuracy of distinguishing four-variable NPN classes was determined to be maximal at 97.8% when 10000 epochs were performed.

**Table 3.** Results of the discrimination learning

	Data size				
	$\times 1$	$\times 2$	$\times 4$	$\times 6$	$\times 8$
Training set accuracy	95.1%	95.3%	94.6%	93.9%	93.3%
Test set accuracy	89.7%	89.2%	85.0%	84.9%	77.6%

Next, we evaluated the networks on test datasets different from the train sets as much as possible. As was expected, the average accuracy on the test set is lower than the discrimination accuracy on the training dataset. We observed that in both cases, the False-Negatives are, in general, two to three times more numerous than the False-Positives. A similar pattern occurs for the smaller dataset. Certain groups of functions are more learnable than others.

## 4 Conclusion

In this paper, we presented the study of approximate Boolean functions classification using machine learning. We demonstrated that the NPN classification

could be perfect when the parameter space is searched systematically. An interesting observation is that one can generate NPN classification for  $n$ -variable function from  $n - 1$ -variable NPN function classes. Finally, we also showed that one can train a classifier for function discrimination without explicit NPN class computation. Some of the classes are easier to train, some tougher. It depends on the size of the class as well as on their representatives. In future works, we will test if these tougher classes also have a more complicated circuit representation.

The usefulness of this approach can be seen in the possible application to logic synthesis. In particular, the learnability of the NPN classes could be used for estimating the cost of logic circuits before synthesis by learning the function to be synthesised. The learning result can be then used to determine the synthesis method. Therefore the future work is to look closer at the individual classes learned and determine the difficulty to learn specific categories such as linear, symmetric, bent, etc.

## References

1. Debnath, D., Sasao, T.: Efficient computation of canonical form under variable permutation and negation for Boolean matching in large libraries. *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.* **E89-A**(12), 3443–3450 (2006)
2. Edwards, C.R.: The application of the Rademacher-Walsh transform to Boolean function classification and threshold logic synthesis. *IEEE Trans. Comput.* **100**(1), 48–62 (1975)
3. Guliyev, N.J., Ismailov, V.E.: Approximation capability of two hidden layer feed-forward neural networks with fixed weights. *CoRR abs/2101.09181* (2021)
4. Harrison, M.A.: On the classification of Boolean functions by the general linear and affine groups. *J. Soc. Ind. Appl. Math.* **12**(2), 285–299 (1964)
5. Hurst, S.L.: *The Logical Processing of Digital Signals*. Crane Russak & Company Inc., Edward Arnold, New York, London (1978)
6. Lukac, M., Moraga, C., Kameyama, M.: Properties of bent functions in the truth domain. In: *2019 International Conference on Information and Digital Technologies (IDT)*, pp. 304–310 (2019)
7. Mhaskar, H., Liao, Q., Poggio, T.A.: Learning real and Boolean functions: when is deep better than shallow. *CoRR abs/1603.00988* (2016)
8. Oliveira, A.L., Sangiovanni-Vincentelli, A.: Learning complex Boolean functions: algorithms and applications. In: *Proceedings of the 6th International Conference on Neural Information Processing Systems (NIPS 1993)*, p. 911–918. Morgan Kaufmann Publishers Inc., San Francisco (1993)
9. Sadohara, K.: Learning of Boolean functions using support vector machines. In: Abe, N., Khardon, R., Zeugmann, T. (eds.) *ALT 2001*. LNCS, vol. 2225, pp. 106–118. Springer, Heidelberg (2001). [https://doi.org/10.1007/3-540-45583-3\\_10](https://doi.org/10.1007/3-540-45583-3_10)
10. Sasao, T.: *Switching Theory For Logic Synthesis*. Cluver Academic Publishers, Boston/London/Dordrecht (1999)
11. Tavares, A.R., Avelar, P., Flach, J.M., Nicolau, M., Lamb, L.C., Vardi, M.: Understanding Boolean function learnability on deep neural networks (2020)
12. Tsai, C.C., Marek-Sadowska, M.: Boolean functions classification via fixed polarity reed-muller forms. *IEEE Tran. Comput.* **46**(2), 173–186 (1997)