Neurocomputing 218 (2016) 276–285

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

Disjunctive normal networks

Mehdi Sajjadi *, Mojtaba Seyedhosseini, Tolga Tasdizen

Electrical and Computer Engineering Department, University of Utah, Salt Lake City, USA

ARTICLE INFO

Article history:
Received 7 March 2015
Accepted 20 August 2016
Communicated by Thimm Georg
Available online 13 September 2016

Keywords:
Supervised learning
Neural networks
Classification

ABSTRACT

Artificial neural networks are powerful pattern classifiers. They form the basis of the highly successful and popular Convolutional Networks which offer the state-of-the-art performance on several computer visions tasks. However, in many general and non-vision tasks, neural networks are surpassed by methods such as support vector machines and random forests that are also easier to use and faster to train. One reason is that the backpropagation algorithm, which is used to train artificial neural networks, usually starts from a random weight initialization which complicates the optimization process leading to long training times and increases the risk of stopping in a poor local minima. Several initialization schemes and pre-training methods have been proposed to improve the efficiency and performance of training a neural network. However, this problem arises from the architecture of neural networks. We use the disjunctive normal form and approximate the boolean conjunction operations with products to construct a novel network architecture. The proposed model can be trained by minimizing an error function and it allows an effective and intuitive initialization which avoids poor local minima. We show that the proposed structure provides efficient coverage of the decision space which leads to state-of-the art classification accuracy and fast training times.

© 2016 Elsevier B.V. All rights reserved.

1. Introduction

An artificial neural network (ANN) consisting of one hidden layer of squashing functions is an universal approximator for continuous functions defined on the unit hypercube [1,2]. However, until the introduction of the backpropagation algorithm [3], training such multilayer perceptron (MLP) networks was not possible in practice. The backpropagation algorithm propelled MLPs to be the method of choice for many classification and regression applications. The success of neural networks culminated by Convolutional Networks [4,5] (ConvNets), which deliver the current state-of-the art performance on many computer vision problems including but not limited to classification, detection, localization and scene labeling [6–10].

However, MLPs are not always the classifier of choice when it comes to general and non-vision tasks. In this regard, other techniques such as support vector machines (SVM) [11] and random forests (RF) [12] are the preferred options. The computational cost of training fully-connected MLPs can be high yet they don't deliver the best accuracy possible especially when the size and dimensionality of the dataset grows. An underlying reason is that the optimization process for training MLPs can become more complicated in higher dimensions. These optimization methods usually start from a random starting point and use the gradient descent to find a locally optimal solution. However, this starting point can be anywhere in the high-dimensional space and possibly in the energy well of a poor local optima. Therefore, it may take the gradient descent a lot of iterations to get to a solution which might possibly be significantly sub-optimal compared to other local minima. This increases the variation in training times. On the other hand, this random initialization, may put the initial point near a local minima and consequently lead the gradient descent to a false local minima solution. Neural networks also suffer from the herd-effect problem [13]. During backpropagation each hidden unit tries to evolve into a useful feature detector from a random initialization; however, this task is complicated by the fact that all units are changing at the same time without any direct communication between them. Consequently, hidden units can not effectively subdivide the necessary computational tasks among themselves leading to a complex dance which can take a long time to settle down.

Another related classification approach is to partition the decision space or cover the desired parts of space and then treat each partition or part of the covered space accordingly. Radial Basis Functions Networks (RBF) are popular examples of these methods. RBFs are commonly used models which use radial functions such as Gaussians as basis functions [14]. Each radially symmetric Gaussian is tuned to respond to a local region of feature space. One important drawback of such models is that their local coverage can

* Corresponding author.
E-mail addresses: mehdi@sci.utah.edu (M. Sajjadi), mseyed@sci.utah.edu (M. Seyedhosseini), tolga@sci.utah.edu (T. Tasdizen).

http://dx.doi.org/10.1016/j.neucom.2016.08.047
0925-2312/© 2016 Elsevier B.V. All rights reserved.

be inefficient when non-local coverage of space is needed [15]. In other words, so many local radial basis functions are needed to cover a non-local part of the decision space sufficiently. The reason is that each of these radial functions only covers a local part of the space. It is also well known that an RBF network suffers from the curse of dimensionality [16]. As the number of dimensions grow, the number of radial basis functions required grows exponentially.

In this paper, we introduce a new network architecture that overcomes the difficulties associated with MLPs and back-propagation for supervised learning. Our model also provides an efficient space coverage which can either be local or non-local. This is done by designing a set of convex polytopes that unlike RBFs are flexible in shape and adaptive in coverage. Our network consists of one adaptive layer of feature detectors implemented by logistic sigmoid functions followed by two fixed layers of logical units that compute conjunctions and disjunctions, respectively. We call the proposed network architecture Logistic Disjunctive Normal Network (LDNN). Unlike MLPs, LDNNs allow for a simple and intuitive initialization of the network weights which avoids the herd-effect. Furthermore, due to the single adaptive layer, it allows larger step sizes in minimizing the error function. Finally, we present results of experiments conducted on 10 binary and 6 multi-class classification problems. We repeated each trial repeatedly and reported the mean, min and max of error rates for each problem in order to consider the variability in the results. LDNNs outperformed MLPs in every case and produced the best accuracy in 11 out of the 16 classification problems in comparison to SVMs and RFs.

2. Related work

Extensive research has been performed to improve the performance of the backpropagation algorithm including batch vs. stochastic learning [17,18], squared error vs. cross-entropy [19] and optimal learning rates [20,21]. Many other practical choices including normalization of inputs, initialization of weights, stopping criteria, activation functions, target output values that will not saturate the activation functions, shuffling training examples, momentum terms in optimization, and optimization techniques that make use of the second-order derivatives of the error are summarized in [22]. More recently, Hinton et al. proposed a Dropout scheme for backpropagation which helps prevent co-adaptation of feature detectors [23]. Despite the extensive effort devoted to making learning MLPs as efficient as possible, the fundamental problems outlined in Section 1 remain because they arise from the architecture of MLPs. There are several initialization and unsupervised pre-training methods proposed to alleviate the herd-effect problem. For example, Contrastive divergence [24,25] can be used to pre-train networks in an unsupervised manner prior to backpropagation. Contrastive divergence has been used successfully to train deep networks. The LDNN model proposed in this paper can be seen as an architectural alternative for supervised learning of one hidden layer ANNs.

The idea of representing classification functions in disjunctive form has been previously explored in the literature. Fuzzy min-max networks [26–28] represent the classification function as the union of axis aligned hypercubes in the feature space. The most important drawback of this model is its limitation to axis aligned decision boundaries which can significantly increase the number of conjunctions necessary for a good approximation. We construct a significantly more efficient approximation by using an union of convex polytopes. Furthermore, fuzzy min-max neural networks employ an adhoc expansion-contraction scheme for learning, whereas we formulate learning as an energy minimization problem. Lu et al. [29] proposed a multi-sieving network that decomposes learning tasks. Lee et al. [30] proposed a disjunctive fuzzy network which is based on prototypes; however, it lacks an objective function and is based on an adhoc training procedure. Similarly, the modular network proposed by Lu and Ito [31] removes the axis aligned hypercube restriction from fuzzy min-max networks; however, their network can not be learned by minimizing a single energy function. Our LDNN model uses differentiable activation functions which makes it possible to optimize the network parameters in an unified manner by minimizing a single energy function. We show that unified training of our classifier results in very significant accuracy advantages over the modular network. Differentiable approximations of min-max functions have been used to construct fuzzy neural network that can be trained using steepest descent [32–35], but these have produced results that are significantly less accurate than state-of-the-art classification techniques. A closely related approach to ours is adaptive mixtures of local experts which uses a gating network to stochastically select the output from a set of feedforward networks [36]. The reader is referred to [37] for a survey of mixture of expert methods. The products of experts approach models complex probability distributions by multiplying simpler distributions is also related [38].

Besides the network approaches discussed in the previous paragraph, the idea of partitioning the decision space and learning simpler decision functions in each partition has been explored. RBFs mentioned in previous section are related to this approach. The set of radial basis functions are usually Gaussians and their parameters can be obtained by unsupervised clustering of the data or fitting a Gaussian mixture model to our data using the EM algorithm [39]. Then, we can use linear regression to obtain a set of linear weights to combine the responses of the basis functions. It is also possible to learn the RBF parameters in an unified manner by back-propagation of the error using chain rule [40]. Mixture discriminant analysis treats each class as a mixture of Gaussians and learns discriminants between the Gaussians [41]. Subclass discriminant analysis also relies on modeling classes as mixtures of Gaussians prior to learning discriminant [42]. Local linear discriminant analysis clusters the data and learns a linear discriminant in each cluster [43]. In these approaches partitioning of the space is treated as a step independent from the supervised learning step. Wang and Saligrama, proposed a more recent approach that unifies space partitioning and supervised learning [44]. While this method is related in concept to our disjunctive learning, in Section 4.3 we show that LDNNs outperform space partitioning by a large margin. Dai et al. proposed an approach which places local classifiers close to the global decision boundary [45]. Toussaint and Vijayakumar propose a products-of-sigmoids model for discontinuously switching between local models [46]. Another approach greedily builds a piecewise linear classifier by adding classifiers in regions of error clusters [47]. Local versions of SVMs have also been explored [48,49]. A specific type of local classification is based on the idea of pairwise coupling between positive and negative examples or clusters is conceptually close to the initialization we propose for our LDNN model. These methods typically employ a clustering algorithm, learning classifiers between pairs of positive and negative clusters found by clustering, finally followed by a combination scheme such as voting to integrate the pairwise classifiers into a single decision [50–56]. The modular network [31] discussed previously also falls into this category.

There are other methods that share similarities with our proposed structure. One recent example proposed by Goodfellow et al. is Maxout Networks [10]. Instead of using an activation function over the output of a single node, they take the maximum output of a group of hidden nodes as the output. Here, the Max operator acts as an activation function. Maxout is similar to our model in a sense that it combines the output of a set of linear functions using Max operator. However, as we explain in Section 3,

sigmoids are important elements of our model which are not present in Maxout. We also propose a very consistent and intuitive initialization scheme for our model. We provide comparisons with Maxout networks and show that they are outperformed by LDNN. Another work similar to our approach is Sum-Product Networks (SPN) [57]. SPNs are probabilistic models that provide tractable inferences. SPN is based on the notion of *network polynomial* and represents unnormalized probability distributions. This leads to a deep structure with interleaved layers of sums and products. SPN is similar to our proposed structure because it uses sum units to mix different submodels. It also uses products that combine features of submodels. However, our approach is different. Unlike SPNs, we use logistic sigmoid functions before the product layer to approximate half-spaces. Then, we use products to form convex polytopes. Sigmoid functions are not present in SPNs but as mentioned earlier, they are crucial components of our approach.

3. Methods

3.1. Network architecture

Consider the binary classification problem $f: \mathbf{R}^n \rightarrow \mathbf{B}$ where $\mathbf{B} = \{0, 1\}$. Let $\Omega^+ = \{\mathbf{x} \in \mathbf{R}^n: f(\mathbf{x}) = 1\}$. Lets approximate Ω^+ as the union of N convex polytopes $\tilde{\Omega}^+ = \cup_{i=1}^{N} \mathcal{P}_i$ where the i'th polytope is the intersection $\mathcal{P}_i = \cap_{j=1}^{M_i} \mathcal{H}_{ij}$ of M_i half-spaces $\mathcal{H}_{ij} = \{\mathbf{x} \in \mathbf{R}^n: h_{ij}(\mathbf{x}) > 0\}$. We can replace M_i with $M = \max_i M_i$ without loss of generality. H_{ij} is defined in terms of its indicator function

$$h_{ij}(\mathbf{x}) = \begin{cases} 1, & \sum_{k=1}^{n} w_{ijk} x_k + b_{ij} \geq 0 \\ 0, & \text{otherwise} \end{cases}, \tag{1}$$

where w_{ijk} and b_{ij} are the weights and the bias term. Any Boolean function $b: \mathbf{B}^n \rightarrow \mathbf{B}$ can be written as a disjunction of conjunctions, also known as the disjunctive normal form [58]. Hence, we can construct the function

$$\tilde{f}(\mathbf{x}) = \bigvee_{i=1}^{N} \underbrace{\left(\bigwedge_{j=1}^{M} h_{ij}(\mathbf{x}) \right)}_{b_i(\mathbf{x})} \tag{2}$$

such that $\tilde{\Omega}^+ = \{\mathbf{x} \in \mathbf{R}^n: \tilde{f}(\mathbf{x}) = 1\}$. Since $\tilde{\Omega}^+$ is an approximation to Ω^+, it follows that $\tilde{f}$ is an approximation to f. Our next step is to provide a differentiable approximation to this disjunctive normal form. First, the conjunction of binary variables $\bigwedge_{j=1}^{M} h_{ij}(\mathbf{x})$ can be replaced by the product $\prod_{j=1}^{M} h_{ij}(\mathbf{x})$. Then, using De Morgan's laws [58] we can replace the disjunction of the binary variables $\bigvee_{i=1}^{N} b_i(\mathbf{x})$ with $\neg \bigwedge_{i=1}^{N} \neg\, b_i(\mathbf{x})$, which in turn can be replaced by the expression $1 - \prod_{i=1}^{N} (1 - b_i(\mathbf{x}))$. Finally, we can approximate the perceptrons $h_{ij}(\mathbf{x})$ with the logistic sigmoid functions

$$\sigma_{ij}(\mathbf{x}) = \frac{1}{1 + e^{-\sum_{k=1}^{n} w_{ijk} x_k + b_{ij}}}. \tag{3}$$

This yields the differentiable approximation to $\tilde{f}$

$$\hat{f}(\mathbf{x}) = 1 - \prod_{i=1}^{N} \left(1 - \underbrace{\prod_{j=1}^{M} \sigma_{ij}(\mathbf{x})}_{g_i(\mathbf{x})} \right), \tag{4}$$

which can also be visualized as a network (Fig. 1). We refer to the proposed network architecture as LDNN. The only adaptive parameters of the LDNN are the weights and biases of the first layer of logistic sigmoid functions. The second layer consists of N soft NAND gates which implement the logical negations of the conjunctions $g_i(\mathbf{x})$ using products. The output layer is a single soft NAND gate which implements the disjunction using De Morgan's law. We will refer to a LDNN classifier which has N NAND gates in the second layer and M discriminants per NAND gate as a $N \times M$ LDNN. Note that other variations of disjunctive normal networks can be constructed by using any classifier that is differentiable with respect to its parameters in place of the logistic sigmoid functions.

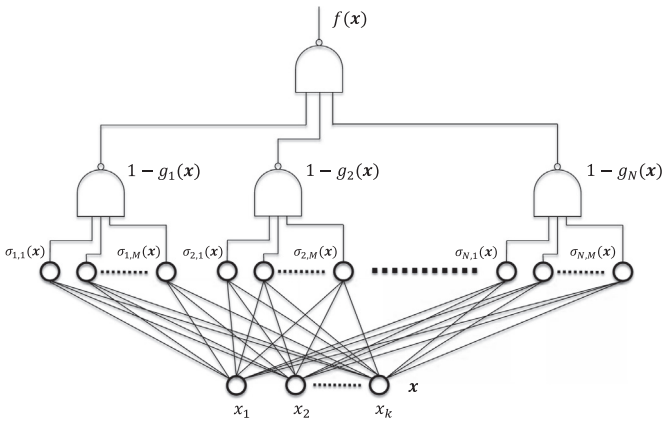

Fig. 1. LDNN architecture. The first hidden layer is composed of $M \times N$ logistic sigmoid functions. The second hidden layer computes the logical negation of N conjunctions using soft NAND gates. The output layer computes the disjunction. The soft NAND gates are implemented as continuous functions by subtracting the product of their inputs from 1.

3.2. Model initialization

Consider a set of training examples $\Gamma = \{(\mathbf{x}, y(\mathbf{x}))\}$ where $y(\mathbf{x})$ denotes the desired binary class corresponding to $\mathbf{x}$. Let Γ^+ and Γ^- be the subsets of Γ for which $y=1$ and $y=0$, respectively. The disjunctive normal form permits a very simple and intuitive initialization of the network weights. To initialize a $N \times M$ LDNN, we first partition Γ^+ and Γ^- into N and M clusters, respectively. Let $\mathbf{v_{ij}} = \mathbf{c}_i^+ - \mathbf{c}_j^-$ where $\mathbf{c}_i^+$ and $\mathbf{c}_j^-$ are the centroids of the i'th positive and j'th negative clusters, respectively. We initialize the weight vectors as $\mathbf{w}_{ij} = \mathbf{v_{ij}}/|\mathbf{v_{ij}}|$. Finally, we initialize the bias terms b_{ij} such that the logistic sigmoid functions $\sigma_{ij}(\mathbf{x})$ take the value 0.5 at the midpoints of the lines connecting the positive and negative cluster centroids. In other words, let $b_{ij} = \langle \mathbf{w}_{ij}, 0.5(\mathbf{c}_i^+ + \mathbf{c}_j^-) \rangle$ where $\langle \mathbf{a}, \mathbf{b} \rangle$ denotes the inner product of the vectors $\mathbf{a}$ and $\mathbf{b}$. This procedure initializes $g_i(\mathbf{x})$, the i'th conjunction in the second hidden layer of the LDNN, to a convex polytope which aims to separate the training instances in the i'th cluster of Γ^+ from all training instances in Γ^-.

We give an intuitive description of LDNN initialization in the context of the two moons dataset. An illustration of this dataset and three clusters for each of the two classes are shown in (Fig. 2a). Initial discriminants for the positive clusters taken one at a time are shown in (Fig. 2b–d). The conjunction of these discriminants form convex polytopes for the positive clusters (Fig. 2e–g). The disjunction of these conjunctions before and after weight optimization (Section 3.3) are illustrated in (Fig. 2h). This initialization procedure is similar to the modular neural network proposed by Lu and Ito (*12*) as well as to locally linear classification by pairwise coupling (*20*) in general. Each module in Lu and Ito's modular network independently learns a linear classifier between a pair of positive and negative training data clusters. The key difference of our classifier from Lu and Ito's network, as well as from

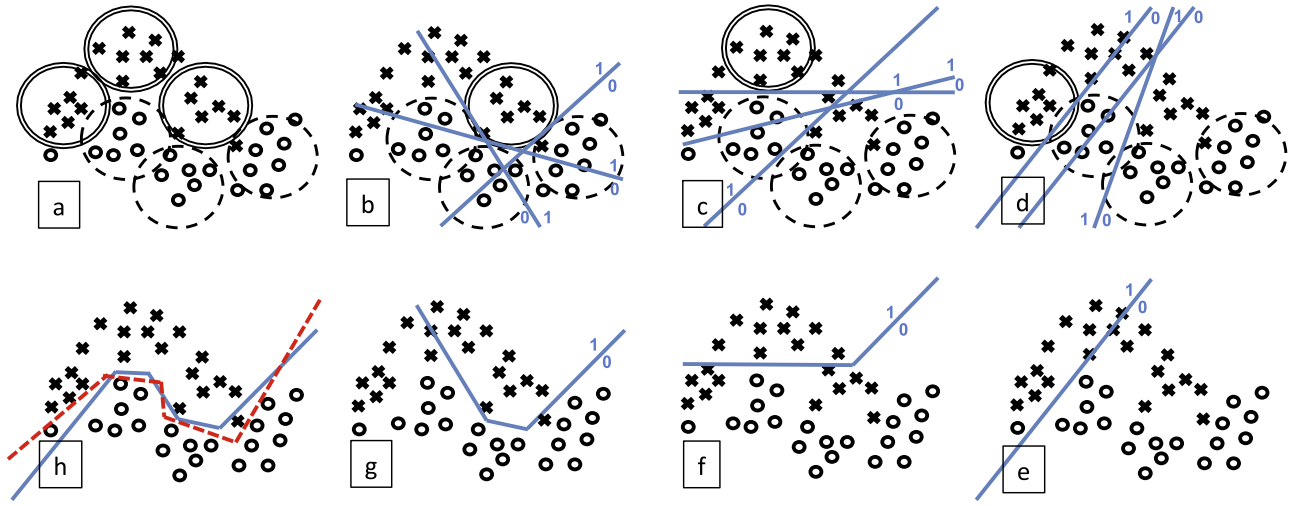

Fig. 2. A binary classification problem: (a) positive and negative training examples partitioned into three clusters each; linear discriminants from each negative cluster to (b) the first positive cluster, (c) the second positive cluster and (d) the third positive cluster; the conjunction of the discriminants for (g) the first positive cluster, (f) the second positive cluster and (e) the third positive cluster; (h) the disjunction of the conjunctions before (blue/solid line) and after (red/dashed line) gradient descent. The 1/0 pair on the sides of the discriminants represent the direction of the discriminant.

locally linear classification by pairwise coupling in general, is that we learn all the linear discriminants simultaneously by minimizing a single error function. When each module is trained independently, the success of the initial clustering can strongly influence the outcome. In Section 4, we show, using both real and artificial datasets, that this important disadvantage can create very significant differences in classification accuracy between modular networks and LDNNs.

3.3. Model optimization

The LDNN model can be trained by choosing the network weights and biases that minimize the quadratic error

$$E(\mathcal{W}, \Gamma) = \sum_{(\mathbf{x},y)\in\Gamma} \left(y - f(\mathbf{x})\right)^2, \tag{5}$$

where f is determined by the set of network weights and biases $\mathcal{W}$. Starting from an initialization as described in Section 3.2, we minimize (5) using gradient descent. To derive the update equations we need to find the partial derivatives of the error with respect to the network weights and biases. Using the fact that $\partial\sigma_{ij}/\partial w_{pqk}$ is non-zero only when $i=p$ and $j=q$, the derivatives of the error function with respect to the network weights is obtained using the chain rule

$$\begin{aligned}\frac{\partial E}{\partial w_{ijk}} &= \frac{\partial E}{\partial f}\frac{\partial f}{\partial g_i}\frac{\partial g_i}{\partial \sigma_{ij}}\frac{\partial \sigma_{ij}}{\partial w_{ijk}} \\ &= -2(y - f(\mathbf{x}))\left(\prod_{r\neq i}(1 - g_r(\mathbf{x}))\right)\times\left(\prod_{l\neq j}\sigma_{il}(\mathbf{x})\right) \\ &\quad \left(\sigma_{ij}(\mathbf{x})(1 - \sigma_{ij}(\mathbf{x}))x_k\right) \\ &= 2(f(\mathbf{x}) - y)\left(\prod_{r\neq i}(1 - g_r(\mathbf{x}))\right)g_i(\mathbf{x})\left(1 - \sigma_{ij}(\mathbf{x})\right)x_k\end{aligned} \tag{6}$$

Similarly, we obtain the derivative of the error function with respect to the network biases as

$$\frac{\partial E}{\partial b_{ij}} = 2(f(\mathbf{x}) - y)\left(\prod_{r\neq i}(1 - g_r(\mathbf{x}))\right)g_i(\mathbf{x})\left(1 - \sigma_{ij}(\mathbf{x})\right) \tag{7}$$

We perform stochastic gradient descent after randomly permuting the order of the instances in Γ and updating the model weights and biases according to $w_{ijk}^{new} = w_{ijk} - \alpha\frac{\partial E}{\partial w_{ijk}}$, and $b_{ij}^{new} = b_{ij} - \alpha\frac{\partial E}{\partial b_{ij}}$, respectively. The constant α is the step size. This constitutes one epoch of training. Multiple epochs are performed until convergence as determined using a separate validation set. Notice that it is possible to achieve 0 training error for any finite training set Γ by letting each positive training instance and each negative training instance represent a positive and negative cluster centroid, respectively. However, in practice, this is expected to lead to overfitting and poor generalization and typically a much smaller number of clusters than training instances is used.

4. Experiments

4.1. Artificial datasets

We first experimented with the *two moons* artificial dataset to evaluate the LDNN algorithm with and without the proposed clustering initialization. We also compare the LDNN model with the modular neural networks(ModN)) [31]. To construct the two moons dataset, we start by generating random radius and angle pairs (r, θ). For both moons, r is an uniform random variable between $R - W/2$ and $R + W/2$ where R and W are parameters that determine the radius and the width of the moons, respectively. For the top moon, θ is an uniformly distributed random variable between 0 and π. For the bottom moon, θ is an uniformly distributed random variable between π and 2π. The Cartesian coordinates for data points on the top and bottom moons are then generated as $(R\cos\theta, R\sin\theta)$ and $(R\cos\theta - W/2, R\sin\theta - \alpha)$, respectively. The parameter α determines the vertical separation between the two moons. We generated a training and a testing dataset by using the parameters $R=1$, $W=0.7$, $\alpha=-0.7$ which generates slightly overlapping classes. Both datasets contained 1000 instances on the top moon and 1000 instances on the bottom moon. Then, for each $n \in [1, 7]$, we trained $50n \times n$ LDNNs starting from random parameter initializations, $50n \times n$ LDNNs initialized from k-means clustering with n clusters per moon and 50 $n \times n$ ModNs initialized from k-means clustering with n clusters per moon. For ModNs, the n^2 linear discriminants are trained independently using data from the n^2 pairs of positive (top moon) and negative (bottom moon) clusters and then combined using min/ma functions. We used stochastic gradient descent with a step size of 0.3, a momentum term weight of 0.1 and 500 epochs for training all models. Testing accuracies

Table 1
Average, min. and max. testing error percentages over 50 repetitions for LDNN initialized with random parameters, initialized with clustering and ModN [31] initialized with clustering for different model sizes.

n	LDNN random init		LDNN cluster init		ModN cluster init	
	Av.	Range	Av.	Range	Av.	Range
1	15.6	[15.2, 18.6]	15.6	[15.2, 20.2]	15.5	[15.2, 16.3]
2	6.6	[3.0, 15.8]	3.3	[2.9, 3.7]	4.2	[3.6, 5.4]
3	4.1	[1.1, 15.6]	2.3	[1.2, 3.5]	2.7	[1.2, 4.8]
4	3.6	[1.2, 15.6]	2.2	[1.3, 3.5]	3.0	[1.8, 5.2]
5	3.4	[1.2, 15.4]	2.2	[1.2, 4.2]	2.8	[1.4, 5.7]

were computed over the second dataset which was not used in training. Table 1 shows the mean, minimum and maximum testing error over the 50 trials for each of the models. We observe that training the LDNN model starting from a random initialization is successful in general; however, the range of testing error rates varies by a larger amount compared to when a cluster initialization is used resulting in a slightly worse mean testing error. We also note that the LDNN model performs better both on average and when comparing the maximum error rates over the 50 trials than the ModN model. Fig. 3 illustrates the output of the LDNN model for $n=3$, which appears to be an appropriate choice based on Table 1. The outputs of the 3 conjunctions are also shown separately to give further intuition into the behavior of the LDNN model. Notice the similarity to Fig. 2(e–h)).

The *two-spirals* dataset is an extremely difficult dataset for the MLP architecture trained with the backpropagation algorithm [13]. The original dataset consists of 194 (x, y) pairs arranged in two interlocking spirals that orbit the origin three times. The classification task is to determine which spiral any given (x, y) point belongs to. We used the farthest distance clustering algorithm [59] for initialization of both models. The k-means clustering algorithm places most centroids near the origin where the data points are denser and fewer centroids on the spiral arms further from the origin where the data is sparser. On the other hand, the farthest distance clustering algorithm provides more uniformly distributed centroids which leads to better classification results with fewer clusters. We performed clustering with maximum distance thresholds 2.2, 2.0 and 1.5 resulting in 18, 21 and 27 clusters per class, respectively. For each of these, we trained a LDNN and a ModN. Note that the number of parameters in both models is the same for the same number of clusters. We used stochastic gradient descent with a step size of 0.3, a momentum term weight of 0.1 and 2000 epochs for training all models. LDNN achieved 0% training error in each of these cases while the ModN's training error was 0.232, 0.062% and 0%, respectively. These results suggest that the unified learning framework of LDNN is able to capture the spiral dataset with many fewer parameters than independent, pairwise learning of discriminants as in [31]. Furthermore, it can be seen from Fig. 4 that LDNN creates a much smoother approximation to the spirals than pairwise learning. Finally, we note that LDNN initialized randomly was not able to find a satisfactory local minimum of the error function via gradient descent. This is similar to the failure of the standard MLP architecture for this dataset. This observation underlines the importance of the existence of an intuitive initialization for the LDNN architecture.

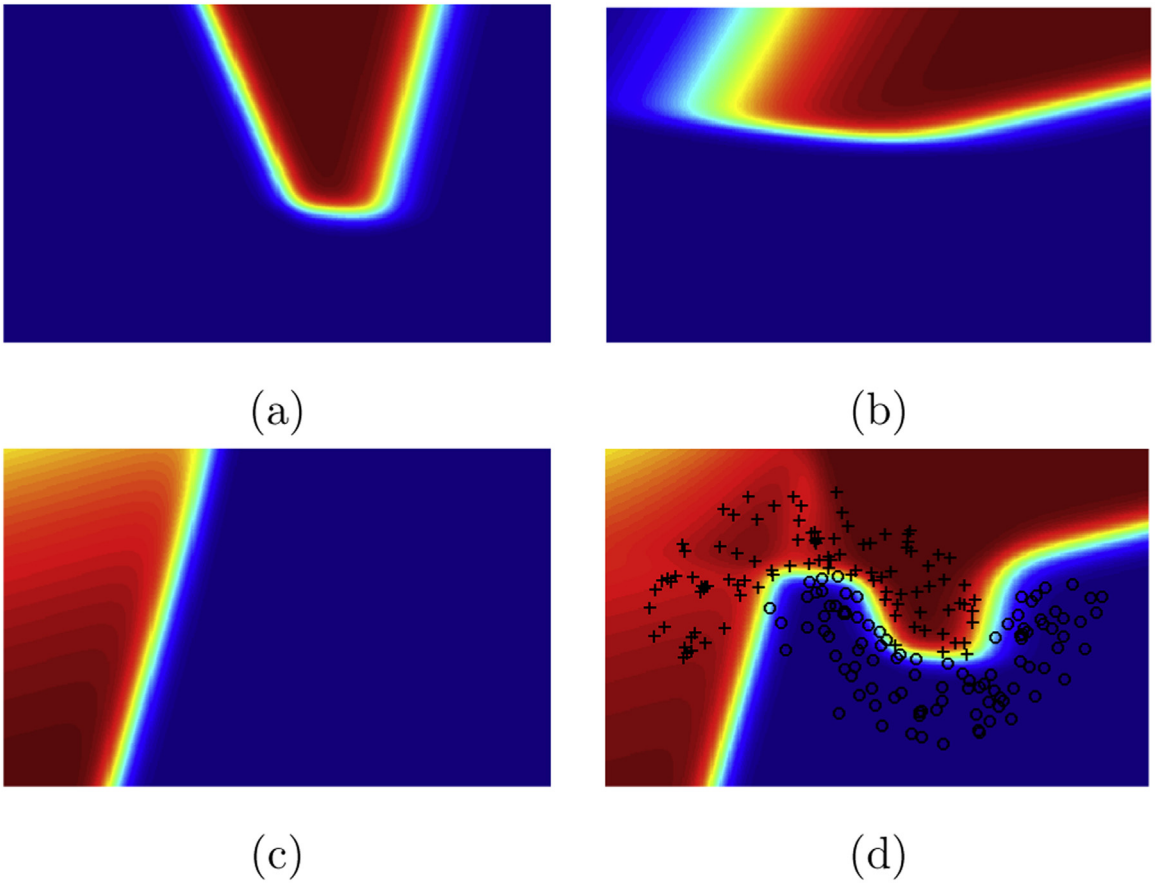

Fig. 3. Two moons test set: (a)–(c) the 3 conjunctions in the second layer of the network evaluated individually, and (d) the output of the 3×3 LDNN. +/o symbols denote the two classes.

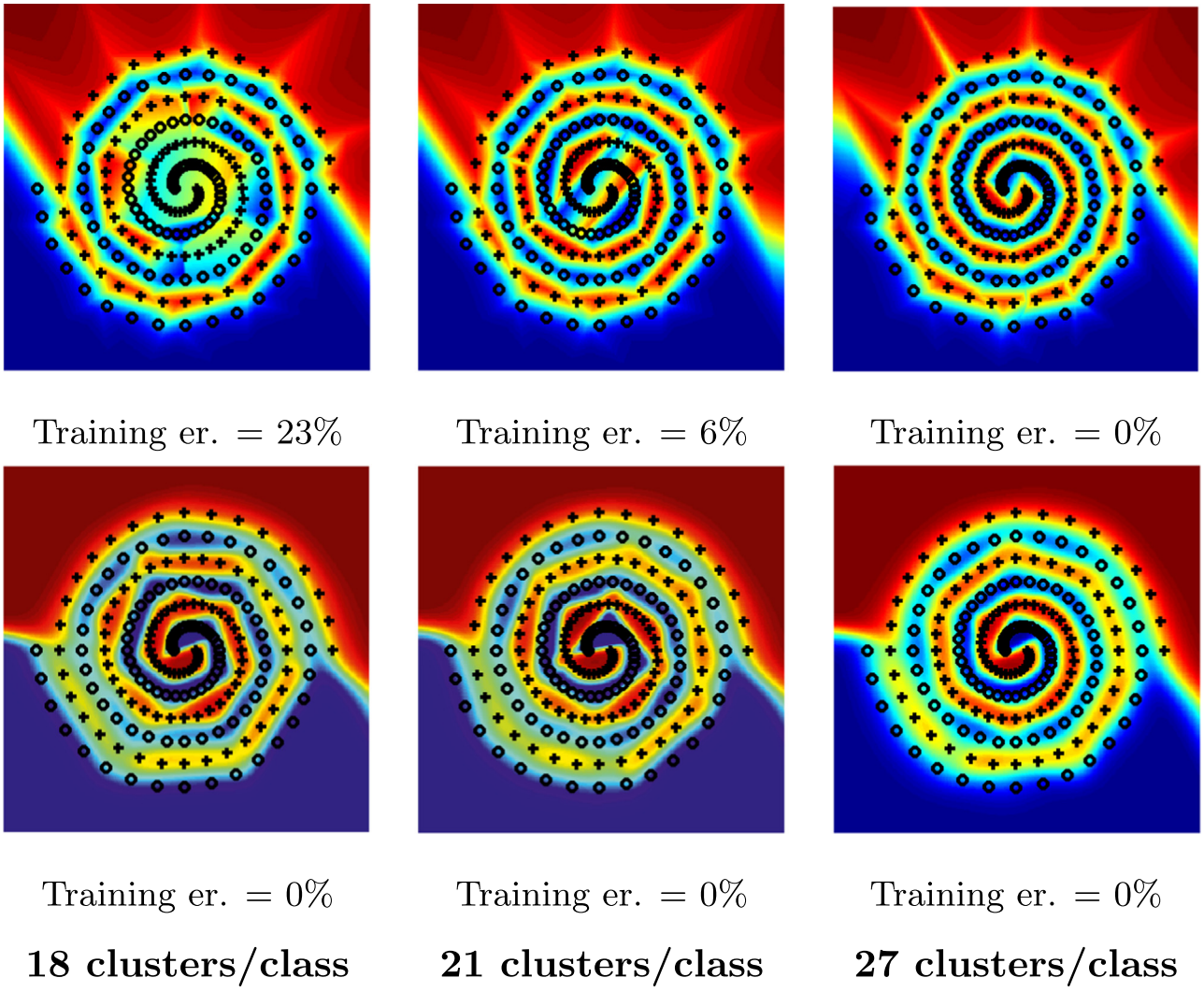

Fig. 4. Two spirals dataset: ModN (top) and LDNN (bottom).

4.2. Two-class problems

We experimented with 10 different binary classification datasets from the *UCI Machine Learning Repository* [60] and the *LIBSVM Tools* webpage [61]. For each dataset, we trained the LDNN, ModN, MLP, SVM, RF, Maxout and RBF classifiers.

4.2.1. Dataset normalization, training/testing set split

Datasets were normalized as follows: For LDNN, ModN, MLP, Maxout and RBF training, we applied a whitening transform [59] to datasets with a large number of training instances (Forest cover type and Webspam) since the covariance matrix could be estimated reliably. All other datasets were normalized by centering each dimension of the feature vector at the origin by subtracting its mean and then scaling by dividing it with its standard deviation. For SVM training, each dimension of the feature vector was linearly scaled to the range [0, 1]. For RF training, no normalization is necessary.

The IJCNN and COD RNA binary datasets had previously determined training and testing sets. For the rest of the datasets, we randomly picked 2/3's of the instances for training and the rest for testing. The training set was further randomly split into a training (%90) and cross-validation (%10) set for determining the parameters of every method.

4.2.2. Model and classifier training parameter selection

For LDNN classifiers we need to choose the number of NAND gates (N) and the number of discriminants per group (M). These parameters translate into the number of positive and negative clusters, respectively in the initialization. Various combinations, up to 40 clusters per class, were tried to find the selection that gives the best accuracy on validation set. For any given number of clusters, the k-means algorithm was repeated 50 times and the clustering result with the lowest sum of square distances to nearest cluster centroid was selected to initialize the LDNN weights. We also fine tuned the step size for gradient descent. The number of epochs for training was selected using the cross-validation set and early stopping. For the IJCNN dataset cross-validation set was also used in training as in [62].

For MLP training, there are two main parameters. The first one is the number of hidden nodes which was varied from 2 to 500 to find the best accuracy on validation set. This was followed by fine tuning the step size for backpropagation. The number of epochs was chosen using the cross-validation set and early stopping. We also trained a second MLP classifier (MLP-m) for which the number of hidden nodes was chosen as $N \times M$ to match the total number of logistic sigmoid functions in the LDNN classifier. This was done to compare LDNN to a MLP with approximately the same degrees of freedom. The optimal parameters for 4 datasets were already matched to LDNN parameters. Similarly, a modular network, which we refer to as Mod-N, with the same number of

conjunctions and disjunctions as the LDNN classifier was trained to control for the degrees of freedom.

There are three main parameters involved in RF training. The first one is the number of trees. We choose a sufficiently large number of trees to ensure that the out of bag error rate converges. The second parameter is the number of features that will be considered in every node of the tree. We tried a range of numbers around the square root of the number of features [12]. The last parameter is the fraction of total samples that will be used in the construction of each tree. We tried 2/3, 1/2, 1/3, 1/4 and 1/5 as possible values for this parameter.

For SVM training, a RBF kernel was used for all datasets except for the MNIST dataset for which a 9th degree polynomial kernel was used [24]. For all datasets except MNIST, we used the grid search tool provided by the Libsvm guide [61] to set the parameters of the RBF kernel.

For Maxout, we need to find the number of hidden nodes h and the number of linear functions per hidden node m. We tried different combinations and picked the one resulting the best performance on validation set. Similar to LDNN and MLP, we fine tuned the step size for gradient descent. The number of epochs for training was also selected using the cross-validation set and early stopping.

RBFs were trained using Netlab [63]. Netlab performs a few steps of k-means to initialize unsupervised learning of a Gaussian Mixture Model using Expectation-Maximization (EM) algorithm. The predicted value is calculated by linearly combining Gaussian kernels. Linear weights are obtained by least squares fitting. For every dataset, we need to find the number of basis functions. So, we tried up to several hundred basis functions to pick the best using cross-validation set.

The training and model parameters selected for all models are listed in Table 2.

4.2.3. Results

All of the classifiers we consider, with the exception of SVM and RBF, are stochastic. Therefore, each experiment with the exception of SVM and RBF was repeated 50 times to obtain mean, minimum and maximum testing errors which are reported in Table 2 for all classifiers. The LDNN classifier outperformed MLPs for all datasets. Furthermore, LDNNs also outperform MLP-m in all datasets. All algorithms were run on an *Intel i7-3770* 3.4 Ghz CPU. Our results signify that the LDNN network architecture and training offers a more accurate alternative to MLPs using back-propagation. The LDNN classifier also outperformed the Mod-N classifier in all datasets including several datasets such as *Forest cover type* and *Wisconsin breast cancer* where the accuracy difference was very large. This emphasizes the importance of training the entire network in an unified manner. Considering all of the classifiers tested, LDNNs had the lowest testing error in 7 out of 10 datasets. LDNNs outperformed SVMs in 8 out of 10 cases, Maxouts in 8 out 10 cases and RFs in 7 out of 10 cases. In 5 out of 10 cases the mean LDNN error was lower than the minimum RF error. The RF mean error was lower than the LDNN minimum error in only 2 out of 10 cases. LDNNs never severely over fit the data, whereas RFs has significant accuracy differences between training and testing sets for several datasets including *Adult*, *PIMA Indian diabetes*, *German credit* and *Forest cover type*. LDNNs outperformed RBFs in all the cases. This can be explained by the way RBF and LDNN cover the decision space. RBFs use simple radial functions to cover desired parts of space. This type of coverage can be inefficient because the basis functions are not flexible and cover only local parts of space. In many cases, a non-local coverage of space is needed which leads to picking too many radial functions in order to cover the space adequately [15]. On the other hand, LDNN combines a set of convex polytopes that are flexible in shape and coverage and can also be local or non-local depending on the type of coverage which is needed.

4.3. Multi-class problems

We also experimented with 6 multi-class datasets from the *UCI Machine Learning Repository* [60]. Each dataset was first normalized in the same way as described in Section 4.2.1. For each dataset we trained the LDNN, RF and SVM classifiers with the exception of the MNIST dataset for which the SVM results are reported from [24]. In that paper, a SVM is trained on a feature set generated by a deep belief network. We used one-vs-all to generalize LDNN to multi-class problems. The model and classifier training parameters were chosen as described in Section 4.2.2 and are reported in Table 3. LDNN and RF experiments were repeated 20 times to obtain mean, minimum and maximum testing errors which are reported in Table 3. The LDNN classifier is also related to the idea of space partitioning [44] which combines partitioning of the space and learning a local classifier for each partition into a global objective function for supervised learning. All space partitioning classifier results are reported from [44]. LDNNs had the best accuracy in 4 out of 6 datasets. Note that the minimum and maximum testing errors for LDNNs were equal for MNIST.

5. Conclusion

We believe that the LDNN network architecture and training can become a favorable alternative to MLPs with one hidden layer. The LDNN classifier has several advantages over MLPs: First, LDNNs allow for a simple and intuitive initialization of the network weights before supervised learning. It is guaranteed that this initialization scheme puts the initial point close to a desired local minima. This makes the optimization process more predictable and leads to more stable and reliable performance. Similarly, LDNN structure along with proposed initialization helps to avoids the herd-effect problem. Second, due to the single adaptive layer, learning can use larger step-sizes in gradient descent. We demonstrated empirically that LDNNs are more accurate than MLPs. Similar to MLPs, the LDNN classifier also requires the choice of model complexity. The number of conjunctions (number of positive training clusters) and the number of logistic sigmoid functions per conjunction (number of negative training clusters) need to be chosen. However, the complexity of the model could be chosen automatically by either using a validation set, as commonly done for SVM training, or by initializing the LDNN in different ways. For instance, sequential covering algorithms can be used to generate a set of rules [64]. Each rule is a conjunction and the final classification is a disjunction of these conjunctions which can easily be converted to a LDNN classifier and fine tuned using gradient descent. LDNNs outperform RBFs significantly which is a proof that LDNNs provide more efficient coverage of the decision space.

While LDNNs are similar in architecture to modular neural networks [31], they are significantly more accurate owing to the unified training of the network that we introduced. LDNNs outperformed RFs in 13 of the 16 datasets and outperformed SVMs in 12 of the 16 datasets. Based on these results and observations, we believe that LDNNs should be considered as a state-of-the art classifier that provides a viable alternative to RFs and SVMs. Further improvements in accuracy can be possible by using cross-entropy instead of the square error criterion or by using adaptive step sizes for training LDNNs. Finally, another possibility is to use more powerful nonlinear discriminants such as conic sections in (3).

Table 2
Column 1: Binary classification datasets, their source, number of positive/negative training/testing examples and data dimensionality. **Column 2**: Classifier type. **Column 3–6**: Average training, average testing, [min, max] testing error (%) and computation time (seconds). Computation times less than 0.1 s are not reported. Best average testing errors are shown in bold. **Column 7**: Model and classifier training parameters used. LDNN, Mod-N: $N \times M$ model and ϵ step size. MLP and MLP-m: h number of hidden nodes and ϵ step size. RF: t number of trees, f number of features considered per node and s training instance sampling rate for each tree. SVM: C penalty factor, γ: RBF kernel width. Maxout: h number of hidden nodes, m number of linear discriminants per hidden node and ϵ step size. RBF: h number of radial basis functions.

Dataset, source and properties	Classifier	Av. train err	Av. test err	Test err range	Time	Model Parameters
Adult	LDNN	15.13	15.25	[15.14, 15.41]	3.1	7×4, ϵ=0.007
UCI	MLP	15.21	15.37	[15.17, 15.74]	2.6	h=20, ϵ=0.005
Train: 7508+/22,654-	RF	7.28	**14.14**	[13.97, 14.30]	9.3	t=300, f=3, s = 2/3
Test: 3700+/11,306-	SVM	15.41	15.57	–	162.7	C=32768, γ=0.007812
Dim=14	Mod-N	17.38	17.39	[16.32, 20.51]	0.9	7×4, ϵ=0.007
	MLP-m	15.26	15.43	[15.14, 15.81]	2.4	h=28, ϵ=0.007
	Maxout	15.01	15.44	[15.14, 15.69]	1.3	h=5, m=4, ϵ=0.005
	RBF	15.86	15.67	–	3.5	h=125
Wisconsin breast cancer	LDNN	1.95	**0.80**	[0.52, 1.58]	<0.1	2×1, ϵ=0.05
UCI	MLP	2.36	1.37	[0.52, 2.64]	<0.1	h=15, ϵ=0.05
Train: 142+/238-	RF	0.32	1.79	[1.58, 2.11]	<0.1	t=300, f=10, s = 2/3
Test: 70+/119-	SVM	2.63	1.59	–	<0.1	C=2048, γ=0.000488
Dim=30	Mod-N	15.58	14.58	[7.93, 24.33]	<0.1	2×1, ϵ=0.05
	MLP-m	2.01	1.59	[0.52, 2.11]	<0.1	h=2, ϵ=0.05
	Maxout	1.44	1.58	[0.52, 3.17]	<0.1	h=2, m=3, ϵ=0.05
	RBF	2.33	1.58	–	<0.1	h=14
PIMA Indians diabetes	LDNN	20.94	**17.92**	[17.25, 19.60]	0.2	6×10, ϵ=0.02
UCI	MLP	24.25	19.34	[16.86, 23.13]	0.2	h=100, ϵ=0.01
Train: 179+/334-	RF	13.20	20.81	[20.39, 21.56]	<0.1	t=150, f=2, s = 1/5
Test: 89+/166-	SVM	21.83	21.57	–	<0.1	C=32, γ=0.125
Dim=8	Mod-N	20.11	24.29	[19.60, 27.05]	<0.1	6×10, ϵ=0.02
	MLP-m	23.51	19.56	[17.64, 22.35]	0.1	h=60, ϵ=0.02
	Maxout	21.84	20.75	[17.64, 23.92]	<0.1	h=6, m=10, ϵ=0.005
	RBF	22.67	18.82	–	<0.1	h=12
Australian credit approval	LDNN	10.04	**12.93**	[12.22, 13.53]	<0.1	5×4, ϵ=0.02
UCI	MLP	11.85	13.90	[12.22, 15.28]	<0.1	h=20, ϵ=0.01
Train: 205+/256-	RF	10.95	12.95	[12.22, 13.10]	<0.1	t=150, f=1, s = 1/5
Test: 1012+/127-	SVM	13.02	16.59	–	<0.1	C=0.03125, γ=0.5
Dim=14	Mod-N	10.43	14.62	[12.22, 17.46]	1.0	5×4, ϵ=0.02
	MLP-m	11.61	14.03	[11.79, 16.15]	<0.1	h=20, ϵ=0.02
	Maxout	12.72	14.15	[12.22, 16.59]	<0.1	h=5, m=3, ϵ=0.005
	RBF	13.94	13.53	–	<0.1	h=3
Ionosphere	LDNN	1.28	**3.40**	[2.56, 4.27]	0.2	1×36, ϵ=0.05
UCI	MLP	1.37	8.66	[6.83, 13.67]	0.2	h=40, ϵ=0.025
Train: 150+/84-	RF	5.42	5.38	[5.12, 5.98]	<0.1	t=200, f=5, s = 1/5
Test: 75+/42-	SVM	0.85	4.27	–	<0.1	C=2, γ=2
Dim=33	Mod-N	1.74	5.98	[4.27, 8.54]	<0.1	1×36, ϵ=0.05
	MLP-m	1.63	8.73	[6.83, 11.11]	<0.1	h=36, ϵ=0.05
	Maxout	0.9	6.37	[2.56, 17.94]	<0.1	h=1, m=36, ϵ=0.05
	RBF	8.05	4.27	–	<0.1	h=21
German credit	LDNN	17.54	**22.58**	[21.02, 23,42]	0.2	6×1, ϵ=0.05
UCI	MLP	1.99	23.96	[22.52, 26.12]	<0.1	h=20, ϵ=0.01
Train: 200+/467-	RF	1.07	24.28	[23.42, 24.92]	0.2	t=250, f=4, s = 2/3
Test: 100+/233-	SVM	11.09	25.83	–	<0.1	C=8, γ=0.125
Dim=24	Mod-N	29.98	30.03	[30.03, 30.03]	<0.1	6×1, ϵ=0.05
	MLP-m	19.16	24.51	[23.12, 26.12]	<0.1	h=6, ϵ=0.05
	Maxout	17.96	25.09	[22.22, 28.22]	<0.1	h=1, m=2, ϵ=0.05
	RBF	21.79	23.72	–	<0.1	h=26
Forest cover type	LDNN	8.22	8.87	[8.09, 9.96]	2702.0	20×10, ϵ=0.1
UCI	MLP	8.01	9.00	[7.73, 13.26]	2790.0	h=200, ϵ=0.1
Train: 188,868+/198,474-	RF	0.29	**3.90**	[3.84, 3.94]	571.1	t=150, f=15, s = 2/3
Test: 94,443+/99,237-	SVM	6.03	6.91	–	13043.0	C=32, γ=8
Dim=54	Mod-N	25.52	25.68	[24.40, 26.77]	55.1	20×10, ϵ=0.1
	Maxout	6.23	7.07	[6.63, 8.13]	5354.8	h=20, m=15, ϵ=0.01
	RBF	24.38	24.53	–	43.2	h=100
IJCNN challenge	LDNN	0.87	**1.28**	[1.02, 1.58]	8.2	10×8, ϵ=0.25
Libsvm	MLP	0.61	1.80	[1.41, 2.27]	19.7	h=80, ϵ=0.1
Train: 3415+ 31,585-	RF	0.08	2.00	[1.91, 2.09]	18.7	t=250, f=3, s = 2/3
Test: 8712+/82,889-	SVM	0.30	1.41	–	38.4	C=32, γ=8
Dim=22	Mod-N	4.68	5.01	[4.13, 7.95]	2.7	10×8, ϵ=0.25
	Maxout	1.10	1.39	[1.03, 3.28]	6.2	h=10, m=10, ϵ=0.1
	RBF	6.96	7.81	–	5.9	h=90
COD-RNA	LDNN	3.59	**3.36**	[3.30, 3.46]	80.9	8×8, ϵ=0.05
Libsvm	MLP	3.14	3.50	[3.37, 3.73]	78.5	h=64, ϵ=0.05
Train: 19,845+/39,690-	RF	0.34	3.37	[3.34, 3.39]	8.3	t=200, f=3, s = 2/3
Test: 90,539+/181,07-	SVM	2.86	3.67	–	157.6	C=512, γ=8
Dim=8	Mod-N	5.29	4.15	[2.82, 4.72]	2.0	8×8, ϵ=0.05
	Maxout	3.91	3.47	[3.35, 3.67]	38.5	h=5, m=10, ϵ=0.05
	RBF	5.41	4.10	–	4.1	h=37

Table 2 (*continued*)

Dataset, source and properties	Classifier	Av. train err	Av. test err	Test err range	Time	Model Parameters
Webspam	LDNN	0.51	1.21	[1.12, 1.27]	401.8	15×15, $\epsilon=0.1$
Libsvm	MLP	0.44	1.25	[1.13, 1.41]	450.0	$h=225$, $\epsilon=0.1$
Train: 141,460+/91,874-	RF	0.02	1.17	[1.13, 1.19]	428.0	$t=100$, $f=11$, $s=2/3$
Test: 70,729+/45,937-	SVM	0.30	**0.78**	–	5345.0	$C=8$, $\gamma=8$
Dim=138	Mod-N	4.52	4.57	[3.89, 5.17]	67.7	15×15, $\epsilon=0.1$
	Maxout	0.12	0.97	[0.9, 1.08]	416.0	$h=12$, $m=17$, $\epsilon=0.02$
	RBF	7.73	7.73	–	167.6	$h=150$

Table 3
Column 1: Multi-class datasets, their source, number of training/testing examples and data dimensionality. **Column 2**: Classifier type. **Column 3–5**: Average training, average testing, and [min, max] testing error (%). Best average testing errors are shown in bold. **Column 6**: Model and classifier training parameters used. LDNN: $N \times M$ model and ϵ step size. RF: t number of trees, f number of features considered per node and s training instance sampling rate for each tree. SVM: C penalty factor, γ: RBF kernel width. The space partitioning (SP) results are from [44].

Dataset, source and properties	Classifier	Av. train err	Av. test err	Test err range	Model parameters
Isolet	LDNN	0.25	4.17	[3.65, 4.49]	4×4, $\epsilon=0.01$
Train: 6238/Test: 1559	RF	0	5.61	[5.25, 5.90]	$t=200$, $f=30$, $s=2/3$
Classes=26, Dim=617	SVM	0	**3.21**	–	$C=8$, $\gamma=0.03125$
	SP-LDA	–	5.58	–	Results taken from[44]
Landsat	LDNN	2.66	**7.98**	[7.65, 8.25]	9×9, $\epsilon=0.1$
Train: 4435/Test: 2000	RF	0.22	9.15	[8.65, 9.55]	$t=200$, $f=6$, $s=2/3$
Classes=6, Dim=36	SVM	1.98	8.15	–	$C=2$, $\gamma=8$
	SP-LDA	–	13.95	–	Results taken from[44]
Letter	LDNN	0.20	**2.32**	[2.12, 2.72]	20×20, $\epsilon=0.4$
Train: 16,000/Test: 4000	RF	0	3.89	[3.65, 4.02]	$t=500$, $f=3$, $s=2/3$
Classes=26, Dim=16	SVM	0.08	2.35	–	$C=8$, $\gamma=8$
	SP-LR	–	13.08	–	Results taken from[44]
Optdigit	LDNN	0.01	2.29	[2.00, 2.67]	5×5, $\epsilon=0.1$
Train: 3823/Test: 1797	RF	0	2.89	[2.50, 3.11]	$t=200$, $f=7$, $s=2/3$
Classes=10, Dim=62	SVM	0.03	**1.56**	–	$C=8$, $\gamma=0.125$
	SP-P	–	4.23	–	Results taken from[44]
Pendigit	LDNN	0.34	**1.80**	[1.68, 1.94]	8×8, $\epsilon=0.005$
Train: 7494/Test: 3498	RF	0.01	3.64	[3.40, 3.83]	$t=250$, $f=4$, $s=2/3$
Classes=10, Dim=16	SVM	0.03	1.86	–	$C=8$, $\gamma=2$
	SP-P	–	4.32	–	Results taken from[44]
MNIST	LDNN	0.03	**1.23**	[1.23, 1.23]	30×30, $\epsilon=0.45$
Train: 60,000/Test: 10,000	RF	0	3.00	[2.88, 3.14]	$t=500$, $f=26$, $s=2/3$
Classes=10, Dim=717	SVM	–	1.40	–	Results taken from[24]

Acknowledgments

This work was supported by NSF Grant IIS-1149299.

References

[1] K. Hornik, M. Stinchcombe, H. White, Multilayer feedforward networks are universal approximators, Neural Netw. 2 (1989) 359–366.
[2] G. Cybenko, Approximation by superpositions of a sigmoidal function, Math. Control Syst. 2 (1989) 303–314.
[3] D. Rumelhart, G. Hinton, R. Williams, Learning Representations by Back-propagating Errors, Nature.
[4] Y. LeCun, B. Boser, J. Denker, D. Henderson, R. Howard, W. Hubbard, L. Jackel, Handwritten digit recognition with a back-propagation network, in: Advances in Neural Information Processing Systems (NIPS 1989), 2, Denver, CO, 1990.
[5] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, Proc. IEEE 86 (11) (1998) 2278–2324.
[6] C. Farabet, C. Couprie, L. Najman, Y. LeCun, Learning hierarchical features for scene labeling, IEEE Transactions on Pattern Analysis and Machine Intelligence.
[7] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, Y. LeCun, Overfeat: Integrated recognition, localization and detection using convolutional networks, in: International Conference on Learning Representations (ICLR2014), 2014.
[8] D. Ciresan, U. Meier, J. Schmidhuber, Multi-column deep neural networks for image classification, in: Computer Vision and Pattern Recognition, 2012, pp. 3642–3649.
[9] L. Wan, M. Zeiler, S. Zhang, Y. LeCun, R. Fergus, Regularization of neural networks using dropconnect, in: Proc. International Conference on Machine learning (ICML'13), 2013.
[10] I. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, Y. Bengio, Maxout networks, in: ICML, 2013.
[11] C. Cortes, V. Vapnik, Support-vector networks, Mach. Learn. 20 (3) (1995) 273–297.
[12] L. Breiman, Random forests, Mach. Learn. 45 (1) (2001) 5–32.
[13] S.E. Fahlman, C. Lebiere, The cascade-correlation learning architecture, in: Advances in Neural Information Processing Systems, 2, Morgan Kaufmann, 1990, pp. 524–532.
[14] D.S. Broomhead, D. Lowe, Radial Basis Functions, Multi-variable Functional Interpolation and Adaptive Networks, Tech. Rep., DTIC Document, 1988.
[15] M. Sajjadi, M. Seyedhosseini, T. Tasdizen, Nonlinear Regression with Logistic Product Basis Networks.
[16] R. Kohn, M. Smith, D. Chan, Nonparametric regression using linear combinations of basis functions, Stat. Comput. 11 (4) (2001) 313–322.
[17] T. Heskes, B. Kappen, Mathematical Approaches to Neural Networks, Elsevier, 1993, Ch. On-line Learning Processes in Artifical Neural Networks.
[18] G. Orr, Dynamics and Algorithms for Stochastic Learning, pH.D. thesis, Oregon Graduate Institute, 1995.
[19] M. Joost, W. Schiffmann, Speeding up backpropagation algorithms by using cross-entropy combined with pattern normalization, Int. J. Uncertain., Fuzziness Knowl.-Based Syst. 6 (2) (1998).
[20] D. Saad, S. Solla, Exact solution for on-line learning in multilayer neural networks, Phys. Rev. Lett. 74 (1995) 4337–4340.
[21] N. Murata, K. Muller, A. Ziehe, S. Amari, Adaptive on-line learning in changing environments, in: Advances in Neural Information Processing Systems, vol. 9, 1997.
[22] Y. LeCun, L.B. a d G Orr, K. Muller, Neural Networks: Tricks of the trade, Springer, 1998, Ch. Efficient BackProp.
[23] G. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, Improving Neural Networks by Preventing Co-adaptation of Feature Detectors, arxiv.org/abs/1207.0580.
[24] G. Hinton, S. Osindero, Y. Teh, A fast learning algorithm for deep belief nets, Neural Comput. 18 (2006) 1527–1554.
[25] G. Hinton, R. Salakhutdinov, Reducing the dimensionality of data with neural networks, Science 313 (5786) (2006) 504–507.
[26] P. Simpson, Fuzzy min-max neural networks - part 1: classification, IEEE Trans. Neural Netw. 3 (5) (1992) 776–786.

[27] B. Song, R.M. II, S. Oh, P. Arabshahi, T. Caudell, J. Choi, Adaptive membership function fusion and annihilation in if-then rules, in: Proc. Int. Conf. Fuzzy Syst., II, 1993, pp. 961–967.
[28] A. Nandedkar, P. Biswas, A fuzzy min-max neural network classifier with compensatory neuron architecture, in: Int. Conf. on Pattern Recognition, 2004.
[29] B.-L. Lu, H. Kita, Y. Nishikawa, A multi-sieving neural network architecture that decomposes learning tasks automatically, in: Proc. Int. Conf. on Neural Networks, 3, 1994, pp. 1319–1324.
[30] H.-M. Lee, K.-H. Chen, I.-F. Jiang, A neural network classifier with disjunctive fuzzy information, Neural Netw. 11 (1998) 1113–1125.
[31] B.-L. Lu, M. Ito, Task decomposition and module combination based on class relations: a modular neural network for pattern classification, IEEE Trans. Neural Netw. 10 (5) (1999) 1244–1256.
[32] R.M. II, S. Oh, P. Arabshahi, T. Caudell, J. Choi, B. Song, Steepest descent adaptations of min-max fuzzy if-then rules, in: Proc Int. Joint Conf. on Neural Networks, III, 1992, pp. 471–477.
[33] H. Normura, I. Hayashi, N. Wakami, A learning method of fuzzy inference by descent method, in: Proc. Int. Conf. Fuzzy Syst., 1992.
[34] X. Zhang, C.-C. Hang, S. Tan, P.-Z. Wang, The delta rule and learning for min-max neural networks, in: Proc. Int. Conf. Neural Networks, 1994, pp. 38–43.
[35] X. Zhang, C.-C. Hang, S. Tan, P.-Z. Wang, The min-max function differentiation and training of fuzzy neural networks, IEEE Trans. Neural Netw. 7 (5) (1996) 1139–1150.
[36] R. Jacobs, M. Jordan, S. Nowlan, G. Hinton, Adaptive mixtures of local experts, Neural Comput. 3 (1991) 79–87.
[37] S. Yuksel, J. Wilson, P. Gader, Twenty years of mixture of experts, IEEE Trans. Neural Netw. Learn. Syst. 23 (8) (2012) 1177–1193.
[38] G. Hinton, Training products of experts by minimizing contrastive divergence, Neural Comput. 14 (8) (2002) 1771–1800.
[39] M.J. Orr, Recent advances in radial basis function networks, Relatório técnico, Centre for Cognitive Science, University of Edinburgh.
[40] F. Schwenker, H.A. Kestler, G. Palm, Three learning phases for radial-basis-function networks, Neural Netw. 14 (4) (2001) 439–458.
[41] T. Hastie, R. Tibshirani, Discriminant analysis by gaussian mixtures, J. R. Stat. Soc. Ser. B 58 (1996) 155–176.
[42] M. Zhu, A. Martinez, Subclass discriminant analysis, IEEE Trans. Pattern Anal. Mach. Intell. 28 (8) (2006) 1274–1286.
[43] T.-K. Kim, J. Kittler, Locally linear discriminant analysis for multimodally distributed classes for face recognition with a single model image, IEEE Trans. Pattern Anal. Mach. Intell. 27 (2005) 318–327.
[44] J. Wang, V. Saligrame, Local supervised learning through space partitioning, in: Advances in Neural Information Processing Systems, 2012.
[45] J. Dai, S. Yan, X. Tang, J. Kwok, Locally adaptive classification piloted by uncertainty, in: Proc. Int. Conf. on Machine Learning, 2006, pp. 225–232.
[46] M. Toussaint, S. Vijayakumar, Learning discontinuities with products-of-sigmoids for switching between local models, in: Proc. Int. Conf. on Machine Learning, 2005, pp. 904–911.
[47] O. Dekel, O. Shamir, There is a hole in my data space: Piecewise predictors for heterogenous learning problems, in: Proc. Int. Conf. on Artificial Intelligence and Machine Learninf, 2012.
[48] H. Cheng, P. Tang, R. Jin, Localized support vector machine and its efficient algorithm, in: Proc. SIAM Int. Conf. on Data Mining, 2007.
[49] T. Hastie, R. Tibshirani, J. Friedman, The Elements of Statistical Learning: Data Mining, Inference and Prediction, Springer, 2001.
[50] B. Schulmeister, F. Wysotzki, Dipol - a hybrid piecewise linear classifier, John Wiley and Sons, 1997, Ch. Machine Learning and Statistics: the Interface, pp. 133–151.
[51] T. Hastie, R. Tibshirani, Classification by pairwise coupling, Ann. Stat. 26 (1) (1998) 451–471.
[52] B. Lu, K. Wang, M. Utiyama, H. Isahara, A part-versus-part method for massively parallel training of support vector machines, in: Proc. of Int. Joint. Conf. on Neural Networks, 2004, pp. 735–740.
[53] T. Wu, C. Lin, R. Weng, Probability estimates for multi-class classification by pairwise coupling, J. Mach. Learn. Res. (2004) 975–1005.
[54] J. Wu, H. Hui, W. Peng, J. Chen, Local decomposition for rare class analysis, in: Proc. ACM Int. Conf. on Knowledge discovery and data mining, 2007, pp. 814–823.
[55] F. Chen, C.-T. Lu, A. Boedihardjo, On locally linear classification by pairwise coupling, in: Data Mining, 2008. ICDM '08. Eighth IEEE International Conference on, 2008, pp. 749–754. http://dx.doi.org/10.1109/ICDM.2008.137.
[56] H. Abbassi, R. Monsefi, H. Sadoghi Yazdi, Constrained classifier: a novel approach to nonlinear classification, Neural Comput. Appl. (2012) 1–11.
[57] H. Poon, P. Domingos, Sum-product networks: a new deep architecture, in: Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on, IEEE, 2011, pp. 689–690.
[58] M. Hazewinkel (Ed.), Encylopedia of Mathematics, Springer, 2001.
[59] R. Duda, P. Hart, D. Stork, Pattern Classification, Wiley-Interscience, 2001.
[60] Uci machine learning repository, archive.ics.uci.edu/ml.
[61] Libsvm, ⟨http://www.csie.ntu.edu.tw/cjlin/libsvmtools/⟩.
[62] C.-C. Chang, C.-J. Lin, Ijcnn 2001 challenge: generalization ability and text decoding, in: International Joint Conference on Neural Networks, 2, 2001, pp. 1031–1036.
[63] Netlab neural network software, ⟨www.aston.ac.uk/eas/research/groups/ncrg/resources/netlab⟩.
[64] T. Mitchell, Machine Learning, McGraw-Hill, 1997.