Neurons with Paraboloid Decision Boundaries for Improved Neural Network Classification Performance

Nikolaos Tsapanos, Anastasios Tefas, *Member, IEEE,* Nikolaos Nikolaidis, *Member, IEEE,* and Ioannis Pitas, *Fellow, IEEE*

***Abstract*—In mathematical terms, an artificial neuron computes the inner product of a d-dimensional input vector $\mathbf{x}$ with its weight vector $\mathbf{w}$, compares it with a bias value w_0 and fires based on the result of this comparison. Therefore, its decision boundary is given by the equation $\mathbf{w}^T\mathbf{x} + w_0 = 0$. In this paper, we propose replacing the linear, hyperplane decision boundary of a neuron with a curved, paraboloid decision boundary. Thus, the decision boundary of the proposed paraboloid neuron is given by the equation $(\mathbf{h}^T\mathbf{x}+h_0)^2 - ||\mathbf{x}-\mathbf{p}||_2^2 = 0$, where $\mathbf{h}$ and h_0 denote the parameters of the directrix and $\mathbf{p}$ denotes the coordinates of the focus. Such paraboloid neural networks are proven to have superior recognition accuracy in a number of applications.**

***Keywords*—Neurons, neural networks, paraboloid surfaces, error back-propagation, non-linear separability, neural network training.**

I. Introduction

THE history of *Artificial Neural Networks* (*ANN*s) is as long as that of machine learning itself, since they both trace their roots in the mid to late 1950's [1]. Originally inspired by the brain's biological neurons [2], the *perceptron* algorithm was developed, in order to train an artificial neuron [3]. However, it was later pointed out that a single neuron cannot learn data that are not linearly separable, a notable example being the *eXclusive OR* (*XOR*) function [4]. In order to overcome this problem, *MultiLayer Perceptrons* (*MLP*s) [5] were developed, in which neurons are arranged in at least 3 layers, where the neurons of each layer are connected to neurons of the following layers. The first layer is called the *input* layer, the last is called the *output* layer and every other layer inbetween is called a *hidden* layer. MLPs have been proven to be universal approximators [6], i.e., capable of approximating any function, given the proper parameters.

MLPs were very popular for several years and some of their applications, such as ALVINN [7], are considered as milestones in machine learning. However, their popularity waned with the emergence of support vector machines. They have seen a recent resurgence, after advances in deep learning [8], [9], [10], recurrent neural networks [11], [12], and extreme learning machines [13], [14]. Neural networks have won several international pattern recognition challenges in the last few years [15], [16].

The basic unit of a neural network is an artificial neuron. Like the biological neurons that inspired its function, a neuron receives an input signal in the form of a vector and calculates the dot product of the input and its weight vectors. The neuron *bias* is added to the sum. At this point, there are several options for an activation (or transfer) function, which determines the neuron output. Common activation functions output a negative or positive signal, based on the sign of the final sum, or a value in $[0, 1]$, or in other intervals.

In this paper, we shall focus on the input to the activation function. The dot product plus bias calculation we just described measures the signed, unnormalized distance of the input vector from a hyperplane, as determined by the weight vector and bias. As such, its decision boundary is a linear hyperplane. This means that such a neuron is particularly suited to discriminate data that are linearly separable (at least locally). As datasets rarely satisfy linear separability, we propose replacing the linear decision boundary with a paraboloid one, so that such a neuron can approximate curved boundaries between different data classes.

In the current bibliography, an alternative to the linear decision boundary is provided by *Radial Basis Function* (*RBF*) neurons [17]. Instead of measuring distance from a hyperplane, RBF neurons measure distance from a point. This results in a hyperspherical or hyperelliptical decision boundary. If a covariance matrix is included in the distance calculation, then the decision boundary can be a rotatable and scalable hyperellipse. However, RBF neural networks are susceptible to overtraining [18].

We propose using a paraboloid [19] decision boundary. Such a boundary can be defined as the locus of points that are equidistant from a *directrix hyperplane* (or *directrix*) and a *focal point* (or *focus*). We will henceforth refer to neurons with hyperplane and paraboloid decision boundaries as *hyperplane neurons* and *paraboloid neurons*, respectively.

Note that the decision boundary of both the RBF and the paraboloid neurons defined in $\mathbb{R}^n$ can be expressed in the general form of a second degree curve, i.e., $\sum_{i,j} a_{ij}x_ix_j + \sum_k b_kx_k + c = 0$, having different parameters a_{ij}, b_k and c. In the case of RBF neurons, they are determined by a positive semidefinite and symmetric matrix. In the case of paraboloid neurons, the parameters of the paraboloid decision surface are

Manuscript received August 3, 2016, revised September 20, 2017 and January 12, 2018, accepted May 9, 2018.

Nikolaos Tsapanos, Anastasios Tefas, Nikolaos Nikolaidis and Ioannis Pitas are with the Department of Informatics, Aristotle University of Thessaloniki, University Campus 54124, Thessaloniki, Greece (e-mail addresses: {niktsap, tefas, nikolaid, pitas}@aiia.csd.auth.gr).

determined by the parameters of the directrix and the focus. Neurons with a completely unrestricted second degree decision boundary can be found in [20]. However, there have been no follow up works on such neurons, to the best of our knowledge.

The main advantage of a paraboloid neuron is its ability to approximate a curved decision surface that would otherwise require multiple hyperplane neurons. While paraboloid neurons and RBF neurons share some disadvantages regarding initialization, paraboloid neurons have the ability to utilize a variation of already available training techniques for hyperplane neurons, in order to overcome the said disadvantages. Furthermore, paraboloid neurons have the ability to approximate a variety of curved surfaces with fewer parameters than the RBF neurons. The ability to approximate curved surfaces with either fewer neurons or fewer parameters per neuron can constitute them particularly attractive to platforms with relatively limited resources, such as the hardware embedded in commercial drones.

Note that the proposed paraboloid neurons should not be confused with parabolic bursting neurons, which are described by the theta biological neuron model [21], and belong to the field of computational neuroscience, rather than machine learning.

This paper is meant as a proof of concept for the superiority of paraboloid neurons over hyperplane ones. The scope is to introduce paraboloid neurons and establish them as a valid alternative to hyperplane neurons, capable of providing improved performance in neural networks due to the increased flexibility of their decision boundary. The case for paraboloid neurons is backed up by evidence through experimental evaluation, meaning that the stated goal is plausible and that this is an endeavor worth pursuing.

For the purposes of this work, we shall mainly study the proposed paraboloid neurons in the context of the single hidden layer network architecture. This helps us keep things relatively simple, without diminishing the value of our contribution to the broader field. As the universal approximator theorem has proven, a single hidden layer is enough for a network to approximate any function to an arbitrarily small margin of error [6]. Single hidden layer architectures were also very popular in the late 80s and early 90s, as indicated by the aforementioned ALVINN [7], which was a self driving vehicle that used this architecture. Furthermore, a multilayer network can be considered as a collection of single layer networks, each receiving preprocessed input [22].

The paper is organized as follows. Section II provides an introduction to hyperplane neurons and neural networks, Section III describes the paraboloid neurons, illustrates their advantages and disadvantages in comparison with hyperplane neurons and discusses the error back-propagation rule for paraboloid neurons. Section IV details a novel training algorithm, which is designed to overcome the disadvantages of paraboloid neurons. Section V discusses the application of paraboloid neurons in Deep Neural Networks. Section VI provides the experimental evaluation of neural networks that include paraboloid neurons and are trained using the proposed novel training algorithms. Section VII concludes the paper.

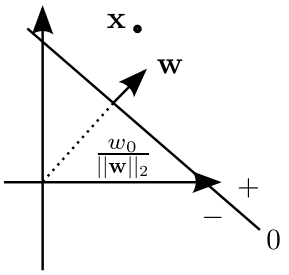


Fig. 1. Hyperplane decision boundary.

II. Hyperplane neurons and neural networks

Let $\mathbf{x}$ be the d-dimensional input vector and $\mathbf{w}$, w_0 be the neuron weight vector and bias, respectively. In correspondence to the biological neurons, high/low positive weights amplify or dampen the relevant input, respectively. The neuron calculates the sum:

$$u = \sum_{i=1}^{d} w_i x_i + w_0 = \mathbf{w}^T \mathbf{x} + w_0, \tag{1}$$

which is the signed, unnormalized distance between a point $\mathbf{x}$ and a hyperplane, whose normal vector is given by $\mathbf{w}$ and whose signed distance from the origin is $\frac{w_0}{||\mathbf{w}||_2}$. The decision boundary of this neuron is the hyper plane defined by $u = 0$ and this is illustrated by a solid line in Figure 1.

An *activation*, or *transfer*, *function* $o = \Phi(u)$, calculates the output o of the neuron. The activation function can be u itself, a simple step function, a hyperbolic tangent, or a sigmoid function [5]. The purpose of using a more complex activation function is to normalize neuron output and to ensure that it is differentiable. For the purposes of this work, we will assume that the sigmoid activation function is employed.

In a neural network, neurons are arranged in *layers*, mainly the *input*, *hidden* and *output* ones. In the feed-forward case, the neurons of each layer are fully connected with the neurons of the next layer and disconnected from any other neuron.

In order to train a neural network by optimizing an error function, the *error back-propagation* algorithm was devised [23], [24], which computes the partial gradient of the error function $E = \frac{1}{2}(||\mathbf{y} - \mathbf{t}||_2)^2$, with respect to every weight, for a given input vector. In this case, $\mathbf{y}$ is the output vector of the output layer of the network and $\mathbf{t}$ is the target vector, i.e., the desired network output given by the ground truth. This is accomplished by initially passing the input vector through the network and recording the output of every neuron, then applying the chain rule to obtain the partial derivative. This partial derivative has been found to be:

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial o_i} \frac{\partial o_i}{\partial u_i} \frac{\partial u_i}{\partial w_{ij}} = \delta_i x_j, \tag{2}$$

where u_i is the dot product of the neurons input vector with its weight vector $\mathbf{w}_i$ and $o_i = \Phi(u_i)$ is the neuron's output. Regarding the parameter δ_i, named the "back-propagation

signal", it has been determined to be:

$$\delta_i = \frac{\partial E}{\partial o_i}\frac{\partial o_i}{\partial u_i} = (y_i - t_i)\Phi(u_i)(1 - \Phi(u_i)), \quad (3)$$

for the output layer and:

$$\delta_i = \frac{\partial E}{\partial o_i}\frac{\partial o_i}{\partial u_i} = (\sum_{k \in \mathcal{L}_{l+1}} \delta_k w_{ki})\Phi(u_i)(1 - \Phi(u_i)). \quad (4)$$

for all other layers.

Note that the output of every neuron is calculated when the input is passed through the network in a feed-forward manner, while the δ of every neuron is calculated backwards. Once the partial derivatives of every neuron have been calculated, we can use *gradient descent* to minimize the error function, by employing the update rule:

$$\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}}, \quad (5)$$

or we can use more sophisticated optimization methods [25], if their application is feasible for the size of the problem.

III. Paraboloid neurons

In this section, we shall describe the function of the proposed paraboloid neurons. We use the definition of a paraboloid as the locus of points that are equidistant from a directrix hyperplane and a focal point. In order to measure distance from a hyperplane, we use the same method as the hyperplane neurons and employ $\mathbf{h}$ and h_0 instead of $\mathbf{w}$ and w_0 to denote the directrix weights and bias, respectively. For an input vector $\mathbf{x}$, we calculate the linear term $\mathbf{h}^T\mathbf{x} + h_0$. Let $\mathbf{p}$ denote the coordinates of the focal point. The distance of point $\mathbf{x}$ from point $\mathbf{p}$ is given by:

$$||\mathbf{x} - \mathbf{p}||_2 = \sqrt{\sum_{i=1}^{d} (x_i - p_i)^2}. \quad (6)$$

Note that $\mathbf{x}, \mathbf{h}, \mathbf{p} \in \mathbb{R}^d$. In order to avoid the squared root and absolute value that would be required to compare these distances directly, we square both of them before comparing them. Thus the signed measure v, which indicates which side of the paraboloid $\mathbf{x}$ falls on is:

$$v = (\mathbf{h}^T\mathbf{x} + h_0)^2 - ||\mathbf{x} - \mathbf{p}||_2^2. \quad (7)$$

If $v < 0$, then $\mathbf{x}$ is closer to the focal point, if $v > 0$, then $\mathbf{x}$ is closer to the directrix and, if $v = 0$, then $\mathbf{x}$ lies exactly on the paraboloid decision boundary. We define the neurons using this equation to be *Type 1* paraboloid neurons. Swapping the terms of the subtraction in Equation (7), i.e., using $v = ||\mathbf{x} - \mathbf{p}||_2^2 - (\mathbf{h}^T\mathbf{x} + h_0)^2$ results in the same decision boundary, but with the signs of v reversed. We define the neurons using this subtraction to be *Type 2* paraboloid neurons. The reason for this distinction will become apparent in Section IV. We will proceed, assuming the paraboloid neurons are Type 1. The Type 2 case is similar, but with the appropriate signs reversed.

Notice that the distance from the directrix should be normalized, i.e., we should use $\frac{\mathbf{h}^T\mathbf{x}+h_0}{||\mathbf{h}||_2}$ instead of $\mathbf{h}^T\mathbf{x} + h_0$. However, we do not normalize this distance, as $||\mathbf{h}||_2^2$ acts as an additional weight to the comparison of these two distances, thus allowing even more flexibility to the decision boundary. Additionally, we do not need to concern ourselves with determining $||\mathbf{h}||_2^2$, as it will be handled by the training algorithm, simply by omitting the normalization step and using (7) directly. Finally, we can also rewrite (7) using only vectors and matrices as so:

$$v = \mathbf{x}^T(\mathbf{h}\mathbf{h}^T - \mathbf{I})\mathbf{x} + 2(h_0\mathbf{h} + \mathbf{p})^T\mathbf{x} + h_0^2 - \mathbf{p}^T\mathbf{p}, \quad (8)$$

where $\mathbf{I}$ is the $d \times d$ identity matrix.

Let us consider the various forms that such a paraboloid decision boundary can take in $\mathbb{R}^d$. If the focal point is infinitely away from the directrix, then the paraboloid becomes a hyperplane. As the two come closer together, the decision boundary becomes a regular paraboloid. If the focus comes very close to the directrix, the decision boundary begins approximating a sharp spike. In the extreme case, where the focus lies on the directrix, the decision boundary becomes a line that goes through the focus and is perpendicular to the directrix. These possibilities are illustrated in Figure 2 for $\mathbb{R}^2$.

Now let us also consider the possibility of appending an extra dimension $x_{d+1} = 1$ at the end of the input vector. In order to visualize this, consider a 2 dimensional flat piece of paper, which will act as the original input space with an additional dimension set to a constant value and a 3 dimensional paraboloid cup, which will act as a paraboloid neuron. The decision boundary in the original input space is determined by the intersection of the cup with the paper. If the focal point of the cup is on the paper and the directrix of the cup is perpendicular to the paper, then their intersection is the same as the one described in the previous paragraph. However, if the cup is placed in such a way that the directrix is on one side of the paper parallel to it and the focus is on the other, then the intersection becomes a circle. If the cup is angled slightly from that position, then the intersection turns from a circle into an ellipse. Such an ellipse can have any rotation possible. This concept is illustrated in Figure 3. Generalizing to the d dimensional space, using a paraboloid neuron in $\mathbb{R}^{(d+1)}$ can provide closed curve decision boundaries, more specifically hyperspheres and hyperellipses. As we can see, the shape of the decision boundary is very flexible. Additionally, in this paper we will focus on paraboloid neurons in a single hidden layer of the neural network, typically but not necessarily the first, as that is the layer that processes the input data in their original space. We assume that any following layers only contain hyperplane neurons, as our formulation of the back-propagation rule includes the δ_i signals of hyperplane neurons, and that the weights of the neurons in any previous layer cannot change any further.

Comparing paraboloid neurons with hyperplane ones, we can see that the former are better suited for approximating curved decision surfaces, while still being able to approximate linear decision boundaries. While it is possible for hyperplane neurons to approximate a curved decision surface, doing so

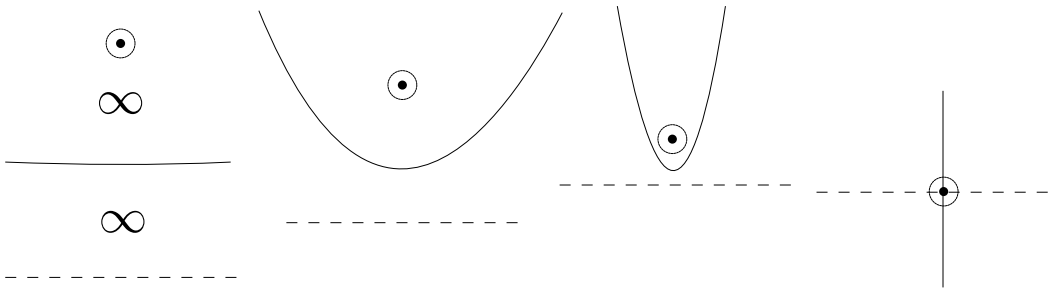


Fig. 2. Paraboloid decision boundary forms. The circle, dashed line and solid line denote the focal point, directrix and decision boundary, respectively.

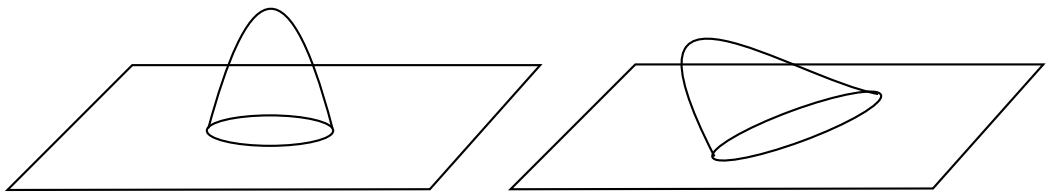


Fig. 3. Paraboloid decision boundary forms in $\mathbb{R}^{(d+1)}$.

will require significantly more neurons than approximating the same surface with paraboloid ones. An example of this can be seen in Figure 4, where the circle is the focal point, the dashed lines are the hyperplane neuron/directrix hyperplanes and the solid lines are the decision curves. RBF neurons provide another alternative to paraboloid neurons. When using the Euclidean distance, RBF neurons can produce hyperspherical decision boundaries and they can also approximate linear decision boundaries, if they are placed extremely far away from the data with an extremely large hypersphere radius. The use of the Mahalanobis distance allows RBF neurons to have fully rotatable hyperelliptical decision boundaries. However, even in this case, paraboloid neurons have the theoretical advantage of being able to have hyperparaboloid decision boundaries, in addition to hyperspherical and hyperelliptical ones. Finally, a Mahalanobis distance RBF neuron has $d^2 + d$ parameters, while a paraboloid neuron in the $(d + 1)$ dimensional space only has $2d + 3$ parameters. Regarding the computational complexities of these neurons, hyperplane neurons require d multiplications and $d+1$ additions, paraboloid neurons require $2d+3$ multiplications and $d+3$ additions, while RBF neurons require $2d^2$ multiplications and $d^2 + 2d$ additions. In practice, the computations for paraboloid neurons take about twice as much time as hyperplane neurons, however, their asymptotic complexity is $O(d)$ in both cases, while it is $O(d^2)$ in the case of RBF neurons.

Unfortunately, for all the flexibility that paraboloid neurons provide, their maneuverability is rather poor. Consider the case illustrated in Figure 5, where the neuron decision surface is curved in the opposite direction of the actual class boundary. In such a case, either the focal point would have to pass through the directrix, or both the focus and the directrix would have to flip for a better fit to the data. However, moving the focus closer to the directrix would increase the curvature of the

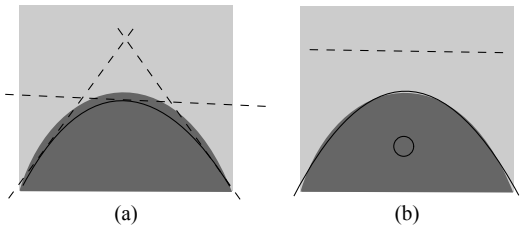


Fig. 4. Comparison between hyperplane neurons and paraboloid neurons. The two grey tones denote two different classes: a) Multiple hyperplane neurons are required to approximate the class boundary. b) A single paraboloid neuron can more closely approximate the same boundary.

decision boundary, thus increasing the error, and flipping them would require very specific and complex changes to neuron parameters. These facts prohibit an optimization method from finding the better solution. In this case, a paraboloid neuron could perform worse than a hyperplane one. We shall handle this issue in a subsequent section with a novel training algorithm, which was specifically devised to circumvent this problem.

We will now adapt the error back-propagation algorithm for paraboloid neurons. Paraboloid neurons are defined by two terms, namely: $H_i = \mathbf{h}_i^T \mathbf{x} + h_{i0}$ and $P_i^2 = \sum_{j=1}^{d} (x_j - p_{ij})^2$. The respective partial derivatives for these terms are given by:

$$\frac{\partial E}{\partial h_{ij}} = \underbrace{\frac{\partial E}{\partial o_i}\frac{\partial o_i}{\partial v_i}}_{\delta_i}\underbrace{\frac{\partial v_i}{\partial H_i^2}}_{1}\underbrace{\frac{\partial H_i^2}{\partial H_i}}_{2H_i}\underbrace{\frac{\partial H_i}{\partial h_{ij}}}_{x_j} = 2\delta_i H_i x_j, \tag{9}$$

$$\frac{\partial E}{\partial p_{ij}} = \underbrace{\frac{\partial E}{\partial o_i}\frac{\partial o_i}{\partial v_i}}_{\delta_i}\underbrace{\frac{\partial v_i}{\partial P_i^2}}_{-1}\underbrace{\frac{\partial P_i^2}{\partial p_{ij}}}_{(-2x_j+2p_{ij})} = 2\delta_i(x_j - p_{ij}), \tag{10}$$

where δ_i is calculated in the same way, as in the traditional back-propagation algorithm.

A. Universal Approximators

We will now prove that paraboloid neural networks are universal approximators. We will use the actual distances from the directrix hyperplane and focal point, instead of their squares. The decision boundaries are theoretically the same and, as we will see, the parameter λ has a bigger effect on the steepness of the activation function than the squaring of distances. Hence, in order to measure the distance of $\mathbf{x}$ from the paraboloid, we will use $v(\mathbf{x}, \mathbf{h}, h_0, \mathbf{p}) = \mathbf{h}^T\mathbf{x} + h_0 - ||\mathbf{x} - \mathbf{p}||_2$.

In order to prove that neural networks that include neurons with paraboloid decision boundaries can approximate any continuous function in the d-dimensional unit hypercube $I_d = [0, 1]^d$, i.e., the universal approximation theorem, we will be following the original proof for feed-forward neural networks [6] almost verbatim, making small changes whenever needed to accommodate paraboloid decision boundaries. In fact, since the paraboloid neurons in the hidden layer are connected to the output layer in a linear fashion, it suffices to prove that a sigmoidal function $\sigma(v(\mathbf{x}, \mathbf{h}, h_0, \mathbf{p}))$ is *discriminatory*, according to the following definitions:

Definition 1. *A function σ is sigmoidal, if:*

$$\sigma(x) \rightarrow \begin{cases} 1 \text{ as } x \rightarrow +\infty, \\ 0 \text{ as } x \rightarrow -\infty, \end{cases}$$

Definition 2. *A function σ is discriminatory, if for a measure $\mu \in M(I_d)$*

$$\int_{I_d} \sigma(v(\mathbf{x}, \mathbf{h}, h_0, \mathbf{p}))d\mu(\mathbf{x}) = 0$$

for all $\mathbf{p}, \mathbf{h} \in \mathbb{R}^d$ and $h_0 \in \mathbb{R}$ implies that $\mu = 0$

Theorem 1. *A function $\sigma(v(\mathbf{x}, \mathbf{h}, h_0, \mathbf{p}))$ is discriminatory.*

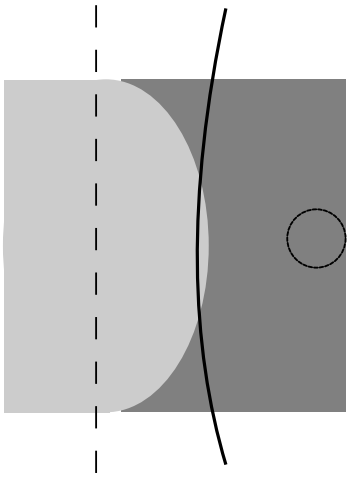


Fig. 5. A paraboloid neuron providing a poor separation of two different classes, illustrated by different grey tones.

Proof: Consider the following function for any $\mathbf{x}, \mathbf{h}, h_0, \mathbf{p}, \phi$:

$$\sigma\left(\lambda(v(\mathbf{x}, \mathbf{h}, h_0, \mathbf{p})) + \phi\right) \begin{cases} \rightarrow 1 \text{ as } \lambda \rightarrow +\infty, \\ \quad \text{for } v(\mathbf{x}, \mathbf{h}, h_0, \mathbf{p}) > 0, \\ \rightarrow 0 \text{ as } \lambda \rightarrow +\infty, \\ \quad \text{for } v(\mathbf{x}, \mathbf{h}, h_0, \mathbf{p}) < 0, \\ \rightarrow \sigma(\phi) \text{ for all } \lambda \\ \quad \text{for } v(\mathbf{x}, \mathbf{h}, h_0, \mathbf{p}) = 0. \end{cases}$$

The functions $\sigma_\lambda(\mathbf{x}) = \sigma\left(\lambda(v(\mathbf{x}, \mathbf{h}, h_0, \mathbf{p})) + \phi\right)$ converge pointwise and boundedly to the function

$$\gamma(\mathbf{x}) = \begin{cases} 1 \text{ for } v(\mathbf{x}, \mathbf{h}, \mathbf{p}, h_0) > 0, \\ 0 \text{ for } v(\mathbf{x}, \mathbf{h}, \mathbf{p}, h_0) < 0, \\ \sigma(\phi) \text{ for } v(\mathbf{x}, \mathbf{h}, h_0, \mathbf{p}) = 0, \end{cases}$$

as $\lambda \rightarrow +\infty$.

Let $\Pi_{\mathbf{h},h_0,\mathbf{p}}$ denote the paraboloid defined by $\{\mathbf{x}|v(\mathbf{x}, \mathbf{h}, h_0, \mathbf{p}) = 0\}$ and $H_{\mathbf{h},h_0,\mathbf{p}}$ denote the interior, open section of the space bisected by the paraboloid and defined by $\{\mathbf{x}|v(\mathbf{x}, \mathbf{h}, h_0, \mathbf{p}) > 0\}$. According to the Lebesgue Bounded Convergence theorem and the assumption of Definition 2, we have that:

$$0 = \int_{I_d} \sigma_\lambda(\mathbf{x})d\mu(\mathbf{x}) = \int_{I_d} \gamma(\mathbf{x})d\mu(\mathbf{x}) = \\ = \sigma(\phi)\mu(\Pi_{\mathbf{h},h_0,\mathbf{p}}) + \mu(H_{\mathbf{h},h_0,\mathbf{p}}).$$

Note that, if $\mathbf{h}^T\mathbf{p} + h_0 < 0$, then $\gamma(\mathbf{x}) = 0$ for all $\mathbf{x}$, the paraboloid $\Pi_{\mathbf{h},h_0,\mathbf{p}}$ and the space $H_{\mathbf{h},h_0,\mathbf{p}}$ do not exist and, therefore, their measures are 0, as $\Pi_{\mathbf{h},h_0,\mathbf{p}} = H_{\mathbf{h},h_0,\mathbf{p}} = \emptyset$ and $\mu(\emptyset) = 0$, thus trivially proving that σ is discriminatory in such a case. We will proceed, assuming that $\mathbf{h}^T\mathbf{p} + h_0 \geq 0$.

Since μ is a signed measure, we must also prove that the measure of all bisected spaces being 0 implies that the measure itself is 0. Fix $\mathbf{p}$ and $\mathbf{h}$. For a bounded measurable function g, define the linear functional F as:

$$F(g) = \int_{I_d} g(||\mathbf{x} - \mathbf{p}||_2 - \mathbf{h}^T\mathbf{x})d\mu(\mathbf{x}),$$

where g is the indicator function of the interval $[h_0, +\infty)$, i.e., $g(u) = 1$, if $u \geq h_0$ and $g(u) = 0$, if $u < h_0$. Note that F is a bounded functional on $L_\infty(\mathbb{R})$, since μ is a finite signed measure [6]. We have:

$$F(g) = \int_{I_d} g(||\mathbf{x} - \mathbf{p}||_2 - \mathbf{h}^T\mathbf{x})d\mu(\mathbf{x}) = \\ = \mu(\Pi_{\mathbf{p},\mathbf{h},-h_0}) + \mu(H_{\mathbf{p},\mathbf{h},-h_0}) = 0.$$

In a similar fashion, $F(g) = 0$, if g is defined as the indicator function of the open interval $(h_0, +\infty)$. By linearity, $F(g) = 0$ for the indicator function of any interval and, as a simple function is a linear combination of indicator functions and other simple functions, $F(g) = 0$ for all simple functions. Since simple functions are dense in $L_\infty(\mathbb{R})$, $F = 0$ [6].

Considering the bounded, measurable functions $c(\mathbf{u}) = \cos(\mathbf{m}^T\mathbf{u})$ and $s(\mathbf{u}) = \sin(\mathbf{m}^T\mathbf{u})$, we have:

$$F(c+is) = \int_{I_d} (\cos(\mathbf{m}^T\mathbf{x}) + i\sin(\mathbf{m}^T\mathbf{x}))d\mu(\mathbf{x}) = \\ = \int_{I_d} e^{(i\mathbf{m}^T\mathbf{x})}d\mu(\mathbf{x}) = 0,$$

for all $\mathbf{m}$, where i denotes the square root of -1. Note that $\int_{I_d} e^{(i\mathbf{m}^T\mathbf{x})}d\mu(\mathbf{x})$ is the Fourier Transform (FT) of measure μ and the FT being 0 means that $\mu = 0$ [6], thus $\sigma(v(\mathbf{x}, \mathbf{h}, h_0, \mathbf{p}))$ is discriminatory. ■

IV. Special paraboloid neuron training algorithm

As we have previously mentioned, paraboloid neurons face issues, when incorrectly initialized, as they may fail to properly separate data classes. In order to acquire a good initialization, instead of using paraboloid neurons from the beginning of the training process, we use auxiliary hyperplane neurons instead. Once the network has reached convergence, we replace the auxiliary neurons with two paraboloid neurons with opposite curvatures and resume training, until the training process converges again.

When presented with a training set that contains n data samples of dimensionality d, we define a $d+1$ dimensional feature vector that contains the values of the original input vector normalized to $[-0.1, 0.1]$ over all input vectors with a 1 appended at the end. We perform the normalization to this interval, instead of the usual $[-1, 1]$, so that we do not have to set the focal point and directrix hyperplane of a paraboloid neuron too far apart, in order to approximate a hyperplanar decision boundary, as this can result in numerical issues. We append a unit at the end of the vector, so that the paraboloid neurons will also be capable of forming closed curve decision boundaries, as discussed in Section III.

The hyperplane neuron provides the initialization for the paraboloid neurons. The aim is to create two paraboloid neurons that approximate the decision boundary of the hyperplane neuron that they will replace, one of which will be Type 1 and the other Type 2, thus preserving the signs of the original neuron. As we have already seen, this is possible, if the focal point is sufficiently enough (ideally infinitely) away from the directrix hyperplane. Since very large numbers cause numerical stability issues, we avoid moving the focus and directrix too far apart from each other. Let c be the constant that represents the offset, by which the focus and directrix will be moved away from the hyperplane neuron. We have found that the value $c = 20$ works well, when all input vector coordinates are normalized to $[-0.1, 0.1]$.

The parameters $\mathbf{h}_1$, h_{10}, $\mathbf{p}_1$ and $\mathbf{h}_2$, h_{20}, $\mathbf{p}_2$ of the first, Type1 and the second, Type 2 paraboloid neuron, respectively, are obtained from the parameters $\mathbf{w}$ and w_0 of the original hyperplane neuron as follows:

$$\mathbf{h}_1 = \frac{\mathbf{w}}{||\mathbf{w}||_2}, h_{10} = \frac{w_0}{||\mathbf{w}||_2} + c, \mathbf{p}_1 = h_{10}\mathbf{h}_1, \\ \mathbf{h}_2 = -\mathbf{h}_1, h_{20} = -h_{10}, \mathbf{p}_2 = h_{20}\mathbf{h}_2.$$

Note that, due to the neuron construction method and the selected value of c, $f_1 = \mathbf{h}_1^T\mathbf{p}_1 + h_{10} = 40$ and $f_2 = \mathbf{h}_2^T\mathbf{p}_2 + h_{20} = 40$, as we have moved the focal point and directrix hyperplane away from the original hyperplane by $c = 20$, in opposite directions. The two paraboloid neurons that replaced the original hyperplane neuron are connected to the neurons of the next layer, with half the value of the original neuron's weight.

After all the auxiliary hyperplane neurons have been replaced by a pair of paraboloid neurons each, as discussed above, we run a single epoch of training. After this epoch has provided us with the updated paraboloid neuron parameters, $\mathbf{h}'_1$, h'_{10}, $\mathbf{p}'_1$, $\mathbf{h}'_2$, h'_{20} and $\mathbf{p}'_2$, we measure $f'_1 = \mathbf{h}'^T_1\mathbf{p}'_1 + h'_{10}$ and $f'_2 = \mathbf{h}'^T_2\mathbf{p}'_2 + h'_{20}$. We set a threshold value θ and we reject the first paraboloid neuron, if $f_1 - f'_1 < \theta$, since this means that the focus and the directrix did not move significantly closer to each other. Therefore, the curvature did not increase. Thus, the neuron decision boundary is probably configured erroneously. Respectively, we reject the second paraboloid neuron, if $f_2 - f'_2 < \theta$. If both paraboloid neurons are rejected, we accept the one with the most curved decision boundary. If both paraboloid neurons are accepted, we can retain the one with the greatest curvature and reject the other one, or we can keep both of them. When a neuron is rejected, the weights of the remaining paraboloid neuron of the pair to the next layer are restored to the values of the original hyperplane neuron. Example cases, in which each of the above scenarios can occur, are illustrated in Figure 6. After we have removed the rejected neurons from the network, we resume training, until it converges. Note that the paraboloid neurons trained with this method should, theoretically, perform at least as well as the hyperplane ones.

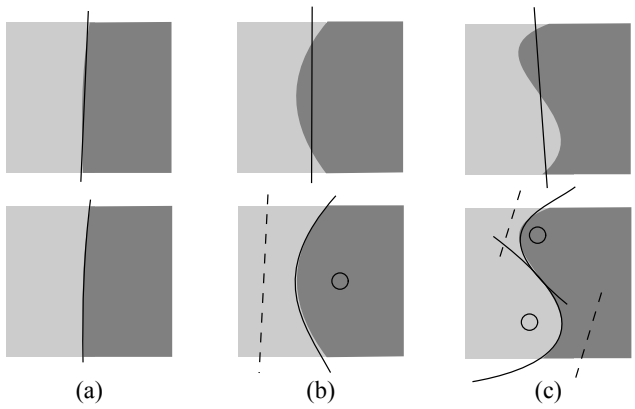


Fig. 6. Various scenarios, in which paraboloid neurons are retained/rejected. Different grey tones denote two different classes. The circle, the dashed line and the solid line denote the paraboloid focal point, directrix and decision boundary, respectively. a) Both neurons would be rejected, because the classes are almost linearly separable. However, one of them is retained instead. b) The neuron which is curved the correct way is retained, while the other is rejected. c) Both neurons are retained.

V. Application in Deep Neural Networks

As we already mentioned, current research trends involve *Deep Neural Networks* (*DNN*s), which include several layers of neurons, sometimes connected in complicated ways. In this section, we will discuss the application of paraboloid neurons in such networks. One key observation that allows us to apply our approach to DNNs is that they usually contain one or more fully connected layers towards the end, so this is where we can use the proposed paraboloid neurons, in an attempt to improve the performance of a DNN. The other key observation is that a DNN can be considered to have two components. One component consisting of the first several layers of the DNN, which can be seen as an adaptive, trainable feature extractor that feeds its output into the second component, consisting of the final two layers, which can be considered as the final classifier. This allows us to treat the first component of a DNN as a black box that maps the input vectors of a dataset unto feature vectors, which can be used to train a single hidden layer paraboloid neural network in the manner we have described. Furthermore, the final two layers of a trained DNN already provide a good initialization for the weights of the converted paraboloid network.

Therefore, we can define the penultimate layer of neurons to use the sigmoid activation function and train a DNN with currently available methods and implementations. Once trained, we can extract the weights of the final two layers of the DNN to initialize a network with paraboloid neurons in the first layer and hyperplane neurons in the output layer, as described in the previous Section. We can then use the output of the previous layer of the DNN as input for the new network with paraboloid neurons and train it further, hoping that it will provide improved results. Note that we require the input to be in $[-0.1, 0.1]^d$, so we need to appropriately scale the input feature vectors and the weights of the hidden layer, in order for the dot product calculated by a neuron to remain unchanged. If $\mathbf{w}$ and w_0 are the hidden layer neuron's weight vector and bias respectively, $\mathbf{D}$ is a diagonal matrix, whose elements d_{ii} are given by $d_{ii} = 0.1/max(|x_{ji}|)$, where x_{ji} is the i-th coordinate of the j-th feature vector, then it is obvious that:

$$\mathbf{w}^T\mathbf{x}_j + w_0 = \underbrace{\mathbf{w}^T\mathbf{D}^{-\mathbf{1}}}_{\text{weights initialization}}\overbrace{\mathbf{D}\mathbf{x}_j}^{\text{scaled vector}} + w_0. \tag{11}$$

The weights between the hidden layer and output layer are extracted exactly as they are. In this paper, we apply this procedure to a *Convolutional Neural Network* (*CNN*), which is a sub-class of deep networks, modeled after biological neuron connectivity that has been found to be particularly suited for Computer Vision related tasks [26], [27]. Now that all the pieces are in place, a synoptic overview of the training process for the proposed paraboloid neural networks can be found in Algorithm 1.

```
if (initialization is available) then
    original-network = initialize from file
else
    original-network = initialize randomly
    for (i = 1 to epochs) do
        shuffle data
        for (j = 1 to n) do
            original-network.feedforward(data[j])
            original-network.backpropagation()
            original-network.updateweights()
        end
        best-network = check for new best
    end
end
paraboloid-network=convert(original-network)
paraboloid-network.optionallycutneurons()
for (i = 1 to epochs) do
    shuffle data
    for (j = 1 to n) do
        paraboloid-network.feedforward(data[j])
        paraboloid-network.backpropagation()
        paraboloid-network.updateparameters()
    end
    best-network = check for new best
end
return best-network
```

Algorithm 1: Paraboloid neural network training algorithm.

VI. Experiments

In this section, we shall present the experimental evaluation of neural networks that use paraboloid neurons. We shall study two cases. The first regards evaluating the impact on classification performance caused by replacing hyperplane neurons with paraboloid neurons in a single hidden layer network that has been trained from the beginning. The second case regards first training a DNN, more specifically a CNN, and then using paraboloid neurons to replace the neurons of the penultimate layer. The converted network is then trained further and the change in classification performance is evaluated. The experimental methodologies and datasets used in each case are described in the beginning of the corresponding subsections. We shall first present the methodology we followed for each experimental evaluation and the results of said experiments. Following that is a discussion, in which we overview experimental results and provide our interpretation of them.

A. Experimental Results

1) Single Hidden Layer: Our basic experimental evaluation methodology for single hidden layer networks is as follows. We begin by defining a training set, a test set and their respective class labels for a given dataset. We first train a hyperplane neural network with one hidden layer and an output layer that contains a number of neurons equal to the number of classes, which we will refer to as the *original hyperplane* network. After training this first network, we train two paraboloid neural networks, as described in section IV, one where we keep all accepted neurons, which will be referred to as *enhanced paraboloid* network and another one, where we only retain one paraboloid neuron for every hyperplane neuron, which is also referred to as *single paraboloid* network. Retaining all the

TABLE I. ERROR RATES OF PROPOSED AND OTHER NEURAL NETWORKS ON THE MNIST HANDWRITTEN DIGITS DATABASE.

Run	Hyperplane						Paraboloid					
	Original			Enchanced			Single			Enhanced		
	Neurons	Training %	Test %	Neurons	Training %	Test %	Neurons	Training %	Test %	Neurons	Training %	Test %
1	300	5.3633%	5.38%	424	5.2650%	5.35%	300	1.8650%	3.48%	424	**1.7350%**	**3.38%**
2	300	5.7267%	5.55%	413	5.3850%	5.51%	300	**1.8183%**	3.69%	413	2.0467%	**3.46%**
3	300	6.3133%	5.96%	430	5.2983%	5.55%	300	1.7517%	3.51%	430	**1.6333%**	**3.19%**
4	300	5.2267%	5.59%	423	5.2317%	5.45%	300	1.9233%	3.56%	423	**1.88%**	**3.30%**
5	300	5.72%	5.71%	409	5.1150%	5.36%	300	1.6850%	3.56%	409	**1.6833%**	**3.22%**
6	300	5.3183%	5.36%	423	5.5533%	5.65%	300	2.1583%	3.41%	423	**1.7683%**	**3.02%**
7	300	5.5467%	5.60%	432	4.9433%	4.92%	300	**1.7417%**	**3.01%**	432	1.7683%	3.25%
8	300	6.14%	5.97%	404	6.3750%	6.44%	300	2.24%	3.82%	404	**1.885%**	**3.44%**
9	300	6.0833%	6.16%	416	5.1583%	5.17%	300	**1.83%**	3.39%	416	1.8383%	**3.33%**
10	300	5.8067%	5.81%	423	5.2917%	5.5%	300	**1.6750%**	**3.13%**	423	1.7467%	3.17%
Mean (STD)	300 (0)	5.72% (0.37%)	5.71% (0.26%)	419.7 (8.9944)	5.36% (0.39%)	5.49% (0.39%)	300 (0)	1.87% (0.19%)	3.46% (0.24%)	419.7 (8.9944)	**1.8% (0.12%)**	**3.28% (0.13%)**

accepted neurons can result in a larger number of neurons, up to double that of the original hyperplane network. We record this number and we then train a final hyperplane network with the increased number of neurons, so as to be fair in our comparisons. We will refer to this last network as *enhanced hyperplane* one. All training is performed using the steepest gradient descent. Finally, we also train some RBF neural networks of similar sizes and compare their performance with both paraboloid networks.

The datasets used in our experiments regarding single hidden layer networks are the following: The MNIST handwritten digit database ($n = 70000$, $d = 784$, 10 classes) [28], the Binghamton University 3D Facial Expression (referred to as BU, $n = 700$, $d = 1200$, 7 classes) [29], the Japanese Female Facial Expression (JAFFE, $n = 210$, $d = 1200$, 7 classes) [30] and the Cohn-Kanade AU-Coded Expression Database (Kanade, $n = 245$, $d = 1200$, 7 classes) [31]. We used the provided training and test sets for the MNIST database, while we used tenfold cross validation for the rest. In the following tables, the "original hyperplane" label refers to the first trained neural network, "enhanced hyperplane" refers to the hyperplane neural network trained with the increased number of neurons. "Single paraboloid" refers to the paraboloid network that only accepted one neuron for each hyperplane and "enhanced paraboloid" refers to the paraboloid network where all accepted neurons were retained. All networks used an input layer, only one hidden layer and an output layer. The replacement of hyperplane neurons with paraboloid neurons was done in the hidden layer. The number of neurons inside the hidden layer is listed in the appropriate entry in the following tables, while the number of neurons in the output layer is equal to the number of classes (10 for MNIST, 7 for the face related datasets). All the neurons used the sigmoid activation function and the optimization objective function was the *Mean Square Error* (*MSE*) measure.

For the MNIST database, we used 300 hyperplane neurons for the original network, as per [28]. We trained the four networks mentioned above 10 times (runs) and the error rates for each network are detailed in Table I. The overall results are presented in the form of *mean* (*standard deviation*) for the classification error. For each run and the mean, the best training and test results are emphasized in bold. Table II presents the comparison between the paraboloid networks and several RBF networks on the MNIST dataset.

TABLE II. TEST ERRORS OF PARABOLOID AND RBF NEURAL NETWORKS IN THE MNIST DATABASE. THE NUMBER IN PARENTHESES AFTER EACH NETWORK LABEL DENOTES THE MEAN NUMBER OF PARABOLOID NEURONS OR THE NUMBER OF RBF NEURONS PER CLASS (10 CLASSES).

Network	Test error %
Paraboloid(300)	3.46% (0.24%)
Paraboloid(419.7)	**3.28% (0.12%)**
RBF(30)	4.69% (0%)
RBF(44)	4.19% (0%)
RBF(50)	4.12% (0%)

For the BU, JAFFE and Kanade face recognition databases, we used a tenfold cross validation approach to evaluate the improvement in performance provided by paraboloid neurons. We initially used 200 hyperplane neurons in all cases. Table III presents the error rates achieved by the hyperplane and paraboloid networks for each face recognition database, while Table IV presents the comparison between the paraboloid networks and the RBF networks for each database. The overall results are presented in the form of *mean* (*standard deviation*) and the best test results are emphasized in bold.

2) Initialization using Convolutional Neural Networks: Our experimental evaluation methodology for studying paraboloid neural networks initialized using a DNN is as follows: We used the MatConvNet [32] example network for the CIFAR-10 dataset [33], which contains a total of 60000 32×32 images for 10 categories of objects. We trained a CNN with MatConvNet, then extracted the feature vectors and the weights of the last two layers, as described in Section V. The penultimate layer of the CNN contained 64 neurons, while the output layer contained 10, which is the number of classes. We then initialized a neural network with the extracted weights and converted it to a paraboloid network, keeping both paraboloid neurons for each hyperplane neuron, resulting in 128 neurons in the hidden layer. Finally, we used the extracted feature vectors, i.e., the scaled input to the penultimate layer of the original CNN for each data sample, in order to train the paraboloid network.

After training several CNNs, we noticed that the training performed by MatConvNet appeared to converge into 4 distinct networks. Additionally, the CIFAR-10 dataset contains 50000 train and 10000 test images and the MatConvNet example uses

TABLE III. ERROR RATES FOR THE FACIAL EXPRESSION RECOGNITION DATABASES.

Database	Hyperplane						Paraboloid					
	Original			Enchanced			Single			Enhanced		
	Neurons	Training %	Test %	Neurons	Training %	Test %	Neurons	Training %	Test %	Neurons	Training %	Test %
BU	200 (0)	0.49% (0.03%)	0.58% (0.04%)	283.4 (6.8670)	0.46% (0.03%)	0.56% (0.04%)	200 (0)	0.21% (0.04%)	0.42% (0.05%)	283.4 (6.8670)	**0.17% (0.04%)**	**0.41% (0.07%)**
JAFFE	200 (0)	0.47% (0.04%)	0.59% (0.08%)	283.05 (6.0912)	0.47% (0.05%)	0.61% (0.09%)	200 (0)	0.12% (0.1%)	0.34% (0.12%)	283.05 (6.0912)	**0.1% (0.08%)**	**0.32% (0.11%)**
Kanade	200 (0)	0.46% (0.04%)	0.6% (0.07%)	285.8333 (7.918)	0.44% (0.06%)	0.6% (0.09%)	200 (0)	0.09% (0.09%)	0.37% (0.13%)	285.8333 (7.918)	**0.07% (0.07%)**	**0.36% (0.13%)**

TABLE IV. TEST ERRORS OF PARABOLOID NEURAL NETWORKS AND RBF NETWORKS IN THE FACIAL EXPRESSION DATABASES. THE NUMBER IN PARENTHESES AFTER EACH NETWORK DENOTES THE MEAN NUMBER OF PARABOLOID NEURONS OR THE NUMBER OF RBF NEURONS PER CLASS (7 CLASSES).

Network	Test error %
BU database	
Paraboloid(200)	0.042% (0.05%)
Paraboloid(283.4)	**0.41% (0.07%)**
RBF(30)	0.5% (0.05%)
RBF(41)	0.49 (0.06%)
RBF(50)	0.47% (0.05%)
JAFFE database	
Paraboloid(200)	0.34% (0.12%)
Paraboloid(283.05)	**0.32% (0.11%)**
RBF(10)	0.38% (0.11%)
RBF(20)	0.42% (0.15%)
RBF(24)	0.43% (0.13%)
Kanade database	
Paraboloid(200)	0.37% (0.13%)
Paraboloid(285.8333)	**0.36% (0.07%)**
RBF(15)	0.52% (0.09%)
RBF(20)	0.51% (0.11%)
RBF(26)	0.53% (0.11%)

the test set as a validation set. For each of the 4 networks, we trained one paraboloid network using the test set as the validation set, for a more direct comparison. Then we also trained another paraboloid network, using only its performance on the training set, so as to test its capability of improving the performance of a very good initial network, without knowledge of the test set. In accordance with the MatConvNet example, the optimization objective function was the *Cross Entropy Loss* (*CEL*) measure, the activation function of the hidden layer neurons was sigmoid, while the output layer neurons used the softmax function.

The results of this experiment are presented in Table V. Since the differences are rather small, the figures in the table indicate raw numbers of errors out of 10000, instead of percentages. The "Starting point" column corresponds to the validation error of the paraboloid network before any training. The "Proposed (validation)" column corresponds to the validation error of the network that used the test set as such. The "Proposed (blind)" column corresponds to the test error of the network trained using the training set and evaluated on the test set, without any knowledge of the latter. Finally, the "Difference" columns contain the change in the performance achieved by the corresponding trained paraboloid network. The overall results are presented in the form of *mean* (*standard deviation*). The best overall performance is emphasized in bold.

B. Discussion

By overviewing the results for the single layer networks, we note that both of the hyperplane neural networks, the original with 300 neurons and the enhanced are with the additional neurons, provide very similar results. This indicates that the test performance of 300 neurons is indeed the limit attained by hyperplane neural networks for this dataset. Paraboloid neural networks, on the other hand, outperform the hyperplane ones in both training and test performance by a significant margin. Additionally, the enhanced paraboloid network, which retained all the accepted neurons, provides a slightly better mean performance and had less error variance than the single paraboloid network, as expected. It should also be noted that the state of the art classification performance for hyperplane neural networks with 300 units on this training set is 4.7% [28], which was surpassed by the proposed paraboloid ones (error rate of 3.28%). Table II illustrates that, while RBF networks outperform hyperplane networks, they are outperformed by the proposed paraboloid networks.

Regarding the BU, JAFFE and Kanade face recognition databases, we can see that the paraboloid neural networks vastly outperform hyperplane ones both in terms of training and test errors. Again, the enhanced paraboloid network outperforms the single paraboloid one in all cases, but one, as can be seen in Table III. An interesting observation is that, in all different datasets, the enhanced paraboloid network used about 40% more neurons. Table IV illustrates that the proposed paraboloid networks again outperform RBF networks.

With respect to the applications of paraboloid neurons in CNNs, the results presented in Table V indicate that the usage of paraboloid neurons can indeed improve upon CNNs. Though the improvement is not very significant, as CNNs are already state-of-the-art competitive, the paraboloid networks consistently provided small, yet measurably better results. In fact, whenever we were able to successfully convert a layer of hyperplane neurons to a layer of paraboloid ones, the resulting networks never failed to provide improvement, thus indicating that paraboloid networks have very strong learning capabilities. Even when the comparison was unfair towards paraboloid neurons, as in the case when they were blind to the test set, there was an average gain in performance when using them, which indicates that paraboloid networks also have strong generalization capabilities.

VII. CONCLUSIONS

In this paper, we have presented the case for paraboloid neural networks, in which neurons have a paraboloid decision

TABLE V. VALIDATION AND TEST ERRORS FOR THE NETWORKS OBTAINED BY CNNS TRAINED USING MATCONVNET AND THE RESULTING PARABOLOID NEURAL NETWORKS ON THE CIFAR-10 DATASET.

Network	Starting point	Proposed (validation)	Difference (validation)	Proposed (blind)	Difference (blind)
1	2009	1984	-25	2002	-7
2	1985	1963	-22	1985	0
3	1961	**1942**	-19	1952	-9
4	1968	1955	-13	1971	3
Mean (STD)	1980.75 (21.36)	1961 (17.6068)	-19.75 (5.1235)	1977.5 (21.2053)	-3.25 (5.6789)

boundary. By replacing the weight vector, bias and dot product involved in the operation of a hyperplane neuron with a combination of two parameter vectors, a bias and a difference of squared distances, we were able to model a paraboloid decision boundary in a simple way that is related to the hyperplane one. It can be trained using the error back-propagation algorithm.

We have shown that the paraboloid neurons are capable of approximating the linear decision boundaries produced by hyperplane neurons. They are also capable of forming closed decision curves and surfaces, i.e., hyperspheres and hyperellipses, like the RBF neurons. Additionally, they are also capable of producing open curve paraboloid decision boundaries, as their namesake suggests. Thus, paraboloid neurons provide the widest variety of possible decision boundaries, with relatively few more parameters than the hyperplane neurons.

Furthermore, we devised a novel training algorithm that uses an already trained hyperplane network as initialization for a paraboloid neural network, by replacing every hyperplane neuron with a pair of paraboloid neurons. This initialization provides a paraboloid network with a good starting point and helps it with avoiding cases in which a paraboloid decision boundary is curved the wrong way rendering its recovery impossible to through error function optimization.

Our experiments indicate that paraboloid neurons are generally useful in improving the performance of neural networks, thus verifying our expectations. Paraboloid neural networks consistently provided better error rates in terms of both the training and test error, than hyperplane ones. Paraboloid neurons even managed to provide an improvement over a state-of-the-art competitive DNN. There are also theoretical guarantees that a paraboloid network will perform at least as well as the network it replaced up to numerical issues.

We believe that paraboloid neurons are powerful tools that have great potential applications in any neural network, including DNNs. We have presented evidence that this is a possibility, as using them in the penultimate layer of a CNN provided improved results. We also believe that even greater potential lies in the possibility of using paraboloid neurons in every layer of a DNN, as even marginal improvements over several hidden layers could snowball into a significant overall boost in performance. In any case, we think that the subject of paraboloid neurons is fresh and unexplored, warranting further investigation.

ACKNOWLEDGEMENT

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731667 (MULTIDRONE). This publication reflects the authors views only. The European Commission is not responsible for any use that may be made of the information it contains.

REFERENCES

[1] B. Farley and W. Clark, "Simulation of self-organizing systems by digital computer," *Transactions of the IRE Professional Group on Information Theory*, vol. 4, no. 4, pp. 76–84, 1954.

[2] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[3] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain," *Psychological review*, vol. 65, no. 6, p. 386, 1958.

[4] M. Minsky and S. Papert, *Perceptrons*, 1969.

[5] C. M. Bishop, *Neural networks for pattern recognition*. Oxford university press, 1995.

[6] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of Control, Signals, and Systems*, vol. 2, no. 4, pp. 303–314, 1989.

[7] D. A. Pomerleau, "ALVINN: an autonomous land vehicle in a neural network," in *Advances in Neural Information Processing Systems 1*, D. S. Touretzky, Ed., 1989, pp. 305–313.

[8] Y. Bengio, A. C. Courville, and P. Vincent, "Representation learning: A review and new perspectives." *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 8, pp. 1798–1828, 2013.

[9] W. Hou, X. Gao, D. Tao, and X. Li, "Blind image quality assessment via deep learning," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 26, no. 6, pp. 1275–1286, 2015.

[10] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Networks*, vol. 61, pp. 85–117, 2015.

[11] M. Lukoeviius and H. Jaeger, "Reservoir computing approaches to recurrent neural network training," *Computer Science Review*, vol. 3, no. 3, pp. 127 – 149, 2009.

[12] J. Hu and J. Wang, "Global stability of complex-valued recurrent neural networks with time-delays," *Neural Networks and Learning Systems, IEEE Transactions on*, vol. 23, no. 6, pp. 853–865, 2012.

[13] G.-B. Huang, D. Wang, and Y. Lan, "Extreme learning machines: a survey," *International Journal of Machine Learning and Cybernetics*, vol. 2, no. 2, pp. 107–122, 2011.

[14] J. Tang, C. Deng, and G. B. Huang, "Extreme learning machine for multilayer perceptron," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 27, no. 4, pp. 809–821, 2016.

[15] A. Graves, M. Liwicki, S. Fernández, R. Bertolami, H. Bunke, and J. Schmidhuber, "A novel connectionist system for unconstrained handwriting recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 31, no. 5, pp. 855–868, 2009.

[16] D. Ciresan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," in *Proceedings of the 25th IEEE Conference on Computer Vision and Pattern Recognition*, 2012, pp. 3642–3649.

[17] V. Kadirkamanathan and M. Niranjan, "A function estimation approach to sequential learning with neural networks," *Neural Computation*, vol. 5, no. 6, pp. 954–975, 1993.

[18] M. Hou and X. Han, "Constructive approximation to multivariate function by decay rbf neural network," *IEEE Transactions on Neural Networks*, vol. 21, no. 9, pp. 1517–1523, 2010.

[19] D. Hilbert and S. Cohn-Vossen, *Geometry and the Imagination*. AMS Chelsea Publishing, 1999.

[20] V. Soon and Y. Huang, "Applications and analysis of second order artificial neural networks," in *Proceedings of the 27th IEEE Conference on Decision and Control*, 1988, pp. 348–349.

[21] G. B. Ermentrout and N. Kopell, "Parabolic bursting in an excitable system coupled with a slow oscillation," *SIAM J. Appl. Math.*, vol. 46, no. 2, pp. 233–253, 1986.

[22] B. Widrow and M. A. Lehr, "30 years of adaptive neural networks: perceptron, madaline, and backpropagation," *Proceedings of the IEEE*, vol. 78, no. 9, pp. 1415–1442, 1990.

[23] P. J. Werbos, "Beyond regression: New tools for prediction and analysis in the behavioral sciences," Ph.D. dissertation, Harvard University, 1974.

[24] P. F. Baldi and K. Hornik, "Learning in linear neural networks: a survey," *IEEE Transactions on Neural Networks*, vol. 6, no. 4, pp. 837–858, 1995.

[25] J. Nocedal and S. Wright, *Numerical optimization*. Springer, 2006.

[26] J. Fan, W. Xu, Y. Wu, and Y. Gong, "Human tracking using convolutional neural networks," *IEEE Transactions on Neural Networks*, vol. 21, no. 10, pp. 1610–1623, 2010.

[27] W. Luo, J. Li, J. Yang, W. Xu, and J. Zhang, "Convolutional sparse autoencoders for image classification," *IEEE Transactions on Neural Networks and Learning Systems*, 2017.

[28] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," in *Proceedings of the IEEE*, 1998, pp. 2278–2324.

[29] L. Yin, X. Wei, Y. Sun, J. Wang, and M. J. Rosato, "A 3d facial expression database for facial behavior research," in *Proceedings of the IEEE International Conference on Face and Gesture Recognition*, 2006, pp. 211–216.

[30] M. Lyons, S. Akamatsu, M. Kamachi, and J. Gyoba, "Coding facial expressions with gabor wavelets," in *Proceedings of the 3rd International Conference on Face & Gesture Recognition*, 1998, pp. 200–205.

[31] T. Kanade, Y. Tian, and J. F. Cohn, "Comprehensive database for facial expression analysis," in *Proceedings of the 4th IEEE International Conference on Automatic Face and Gesture Recognition*, 2000, pp. 46–53.

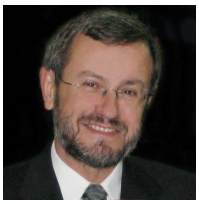
[32] A. Vedaldi and K. Lenc, "Matconvnet: Convolutional neural networks for matlab," in *Proceedings of the 23rd ACM International Conference on Multimedia*, 2015, pp. 689–692.

[33] A. Krizhevsky, "Learning multiple layers of features from tiny images," Tech. Rep., 2009.

Nikolaos Tsapanos received his B.Sc. in Computer Science in 2003 from the University of Ioannina, his M.Sc. in Computer Science, specialized in Technology and Applications in 2005 from the University of Ioannina and his Ph.D. in 2016 from the Aristotle University of Thessaloniki. His current research interests include machine learning, pattern recognition and the inner workings of neural networks.

Anastasios Tefas received the B.Sc. in informatics in 1997 and the Ph.D. degree in informatics in 2002, both from the Aristotle University of Thessaloniki, Greece. Since 2013 he has been an Assistant Professor at the Department of Informatics, Aristotle University of Thessaloniki. From 2008 to 2012, he was a Lecturer at the same University. From 2006 to 2008, he was an Assistant Professor at the Department of Information Management, Technological Institute of Kavala. From 2003 to 2004, he was a temporary lecturer in the Department of Informatics, University of Thessaloniki. From 1997 to 2002, he was a researcher and teaching assistant in the Department of Informatics, University of Thessaloniki. Dr. Tefas participated in 15 research projects financed by national and European funds. He has co-authored 69 journal papers, 177 papers in international conferences and contributed 8 chapters to edited books in his area of expertise. Over 3250 citations have been recorded to his publications and his H-index is 29 according to Google scholar. His current research interests include computational intelligence, pattern recognition, statistical machine learning, digital signal and image analysis and retrieval and computer vision.

Nikolaos Nikolaidis (S92M05SM09) received the Diploma of Electrical Engineering and the Ph.D. degree in Electrical Engineering from the Aristotle University of Thessaloniki, Thessaloniki, Greece, in 1991 and 1997, respectively. He is currently Associate Professor at the Department of Informatics, Aristotle University of Thessaloniki. He has co-authored 1 book, 15 book chapters, 59 journal papers and 177 conference papers and co-edited one book and two special issues in journals. Moreover he has co-organized 6 special sessions in international conferences. The number of citations to his work by third authors exceeds 5300 (h-index 31). He has participated into 24 research projects funded by the EU and national funds. His current areas of interest include anthropocentric and multiview video analysis, analysis of motion capture data, computer vision, digital image/video processing, computer graphics. Dr. Nikolaidis is currently serving as associate/area editor for Signal Processing: Image Communication, EURASIP Journal on Image and Video Processing and IET Image Processing. He served as Technical Program chair of IEEE IVMSP 2013 workshop, and Publicity co-chair of EUSIPCO 2015. He is Publicity co-chair of IEEE ICIP 2018. Dr Nikolaidis is a Senior Member of IEEE.



Prof. Ioannis Pitas (IEEE fellow, IEEE Distinguished Lecturer, EURASIP fellow) received the Diploma and Ph.D. degree in Electrical Engineering, both from the Aristotle University of Thessaloniki, Greece. Since 1994, he has been a Professor at the Department of Informatics of the same University. He served as a Visiting Professor at several Universities. His current interests are in the areas of image/video processing, intelligent digital media, machine learning, human centered interfaces, affective computing, computer vision, 3D imaging and biomedical imaging. He has published over 850 papers, contributed in 44 books in his areas of interest and edited or (co-)authored another 11 books. He has also been member of the program committee of many scientific conferences and workshops. In the past he served as Associate Editor or co-Editor of eight international journals and General or Technical Chair of four international conferences. He participated in 69 R&D projects, primarily funded by the European Union and is/was principal investigator/researcher in 41 such projects. He has 27500+ citations to his work and h-index 80+.