RC25510 (WAT1411-056) November 19, 2014
Mathematics

IBM Research Report

Some Insights into the Geometry and Training of Neural Networks

E. van den Berg
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598
USA

Research Division
Almaden – Austin – Beijing – Cambridge – Dublin - Haifa – India – Melbourne - T.J. Watson – Tokyo - Zurich

Some Insights into the Geometry and Training of Neural Networks

E. van den Berg
IBM T.J. Watson Research Center, USA

June 2, 2014

Abstract

Neural networks have been successfully used for classification tasks in a rapidly growing number of practical applications. Despite their popularity and widespread use, they are often still treated as enigmatic black boxes whose inner workings are insufficiently well understood. In this paper we provide new insights into training and classification by analyzing neural networks from a feature-space perspective. We explain the formation of decision regions and study some of their combinatorial aspects. We place a particular emphasis on the connections between the neural network weight and bias terms and properties of decision boundaries and other regions that exhibit varying levels of classification confidence. We show how the error backpropagates in these regions and emphasize the important role they have in the formation of gradients. These findings expose the connections between scaling of the weight parameters and the density of the training samples. This sheds more light on the vanishing gradient problem, explains the need for regularization, and suggests an approach for subsampling training data to improve performance.

1 Introduction

Neural networks have been successfully used for classification tasks in applications such as pattern recognition [2], speech recognition [9], and numerous others [24]. Despite their widespread use, our understanding of neural networks is still incomplete, and they are therefore often remain treated as black boxes. In this paper we provide new insights into training and classification by analyzing neural networks from a feature-space perspective. We consider feedforward neural networks in which input vectors $x_0 \in \mathbb{R}^d$ are propagated through n successive layers, each of the form

$$x_k = \nu_k(A_k x_{k-1} - b_k), \tag{1}$$

where ν_k is a nonlinear activation function that acts on an affine transformation of the output x_{k-1} from the previous layer, with weight matrix A_k and bias vector b_k. Neural networks are often represented as graphs and the entries in vectors x_k are therefore often referred to as nodes or units. There are three main design parameters in a feedforward neural network architecture: the number of layers or depth the network, the number of nodes in each layer, and the choice of activation function. Once these are fixed, neural networks are training by adjusting only the weight and bias terms.

Although most of the results and principles in this paper apply more generally, we predominantly consider neural networks with sigmoidal activation functions that are convex-concave and differentiable. To keep the discussion concrete we focus on the symmetrized version of the logistic function that acts elementwise on its input as

$$\sigma_\gamma(x) = 2\ell_\gamma(x) - 1, \qquad \text{with} \qquad \ell_\gamma(x) = \frac{1}{1+e^{-\gamma x}}. \tag{2}$$

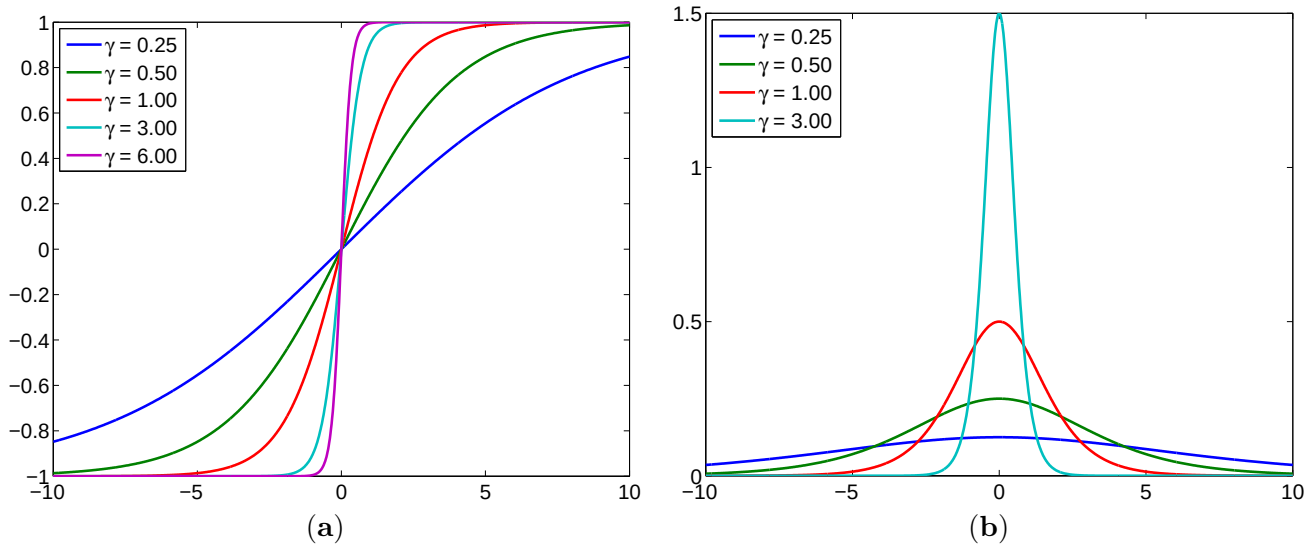


Figure 1: Different instances of (a) the sigmoid function σ_γ and (b) their derivatives.

This function can be seen as a generalization of the hyperbolic tangent, with $\sigma_\gamma(x) = \tanh(\gamma x/2)$. We omit the subscript γ when $\gamma = 1$, or when its exact value does not matter. For simplicity, and with some abuse of terminology, we refer to σ_γ as the sigmoid function, irrespective of the value of γ. Examples of several instances of σ_γ and their first-order derivatives are plotted in Figure 1.

The activation function in the last layer has a special dual purpose of ensuring that the output of the neural network has a meaningful interpretation. The softmax function is widely used and generates an output vector whose entries are defined as

$$[\mu(x)]_i = \frac{e^{x_{[i]}}}{\sum_{j=1}^{k} e^{x_{[j]}}}, \tag{3}$$

Exponentiation and normalization ensure that all output value are nonnegative and sum up to one, and the output of node i can therefore be interpreted as an estimate of the posterior probability $p(\text{class} = i \mid x)$. That is, we can define the estimated probabilities as $\hat{p}_s(\text{class} = i \mid x) := [x_n(x)]_i$, where s is a vector containing of network weight and bias parameters, and $x_n(x)$ is the output at layer n corresponding to input $x_0 = x$. The network parameters s are typically learned from domain-specific training data. In supervised training for multiclass classification this training data comes in the form of a set of tuples $\mathcal{T} = \{(x, c)\}$, each consisting of a sample feature vector $x \in \mathbb{R}^d$ and its associated class label c. Training is done by minimizing a suitably chosen loss function, such as

$$\underset{s}{\text{minimize}} \quad \phi(s) := \frac{1}{|\mathcal{T}|} \sum_{(x,c)\in\mathcal{T}} f(s; x, c). \tag{4}$$

with the cross-entropy function

$$f(s; x, c) = -\log \hat{p}_s(c \mid x),$$

which we shall use throughout the paper. We denote the class c corresponding to feature vector $x \in \mathbb{R}^d$ as $c(x)$, which, in practice, is known only for all points in the training set. For notational convenience we also write $f(x)$ to mean $f(s; x, c(x))$. The loss function $\phi(s)$ is highly nonconvex in s making (4) particularly challenging to solve. Even if it could be solved, care needs to be taken not to overfit the data to ensure

that the network generalized to unseen data. This can be achieved, for example, through regularization, early termination, or by limiting the model capacity of the network.

The outline of the paper is as follows. In Section 2 we review the definition of halfspaces and the formation of decision regions. In Section 3 we look at combinatorial properties of the decision regions, their ability to separate or approximate different classes, and possible generalizations. Section 4 analyzes the connection between the decision regions and the gradient with respect to the different network parameters. Topics related to the training of neural networks including backpropagation, regularization, the contribution of individual training samples to the gradient, and importance sampling are discussed in Section 5. We conclude the paper with a discussion and relevant future work in Section 6.

Throughout the paper we use the following notational conventions. Matrices are indicated by capitals, such as A for the weight matrices; vectors are denoted by lower case roman letters. Sets are denoted by calligraphic capitals. Subscripted square brackets denote indexing, with $[x]_j$, $[A]_i$, and $[A]_{i,j}$ denoting respectively the j-th entry of vector x, the i-th row of A as a column vector, and the (i,j)-th entry of matrix A. Square brackets are also used to denote vector or matrix instances with commas separating entries within one row, and semicolon separating rows in in-line notation. When not exponentiated e denotes the vector of all ones. The largest singular value of A is denoted $\sigma_{\max}(A)$, from the context it will be clear that this is not a particular instance of the sigmoid function σ_γ. The vector ℓ_1, and ℓ_2 norms refer to the one- and two norms; that is, the sum of absolute values and the Euclidean norm, respectively.

2 Formation of decision regions

Decision regions can be described as those regions or sets of points in the feature space that are classified as a certain class. Classification in neural networks is soft in the sense that it comes as a vector of posterior probabilities and it is therefore not immediately obvious how to assign points to one class or another, or if, indeed, this is desirable. Two possible definitions of decision regions for class c are the set of points where the posterior probability is highest among the classes:

$$\mathcal{C}_c := \{x \in \mathbb{R}^d \mid \hat{p}_s(c \mid x) = \max_j \hat{p}_s(j \mid x)\},$$

or exceeds a given threshold:

$$\mathcal{C}_c := \{x \in \mathbb{R}^d \mid \hat{p}_s(c \mid x) \geq \tau\}. \tag{5}$$

Although this section discusses the formation and role of decision regions and its boundaries, we will not use any formal definition of decision regions. However, the intuitive notion used closely follows definition (5).

As we will see in this section, decision regions are formed as input is propagated through the network. Even though the form (1) of all the layers is identical, we can nevertheless identify two distinct stages in region formation. The first stage defines a collection of halfspaces and takes place in the first layer of the network. The second stage takes place over the remaining layers in which intermediate regions are successively combined to form the final decision regions, starting with the initial set of halfspaces. The generation of halfspaces or hyperplanes in the first layer of the neural network and their combination in subsequent layers is well known (see for example [2, 15]). The formation of soft decision boundaries and some of their properties does not appear to have been studied widely. Some of the notions discussed next form the basis for subsequent sections, and we therefore review the two separate stages mentioned above in some detail, with a particular emphasis on the role of the sigmoidal activation function.

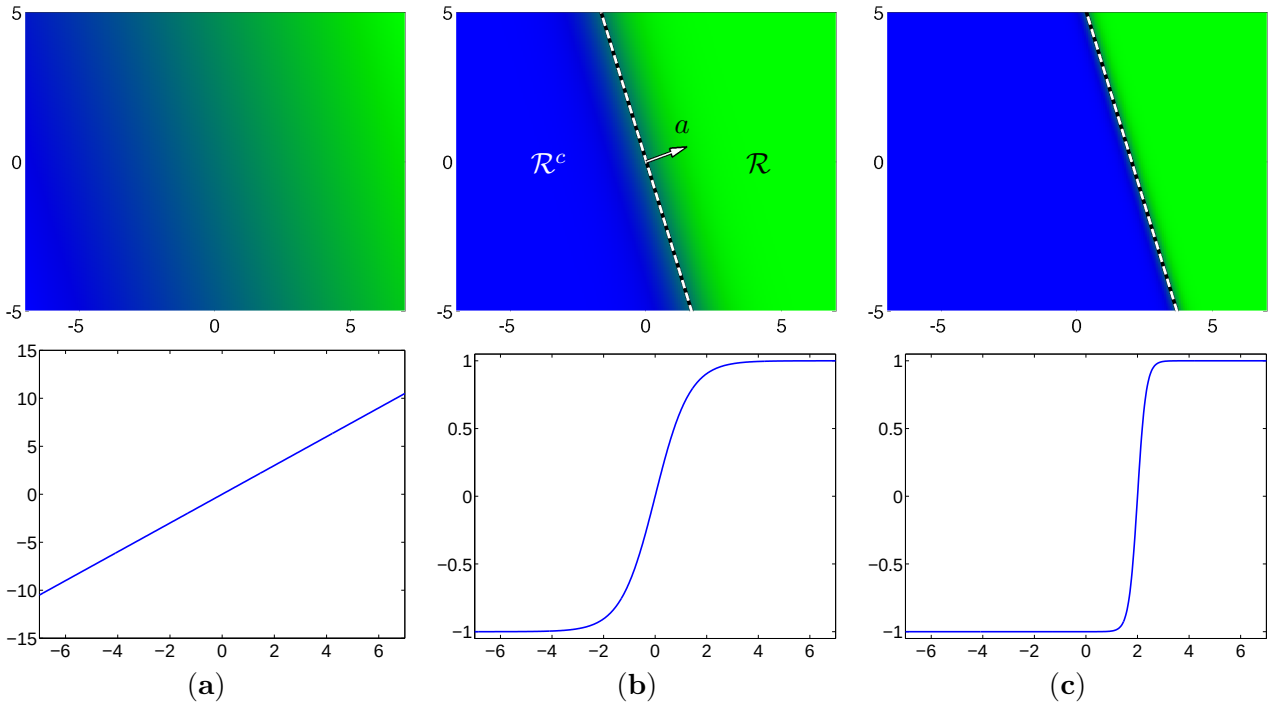


Figure 2: The mapping of points x in the feature space using (a) the linear transformation $\langle a, x\rangle - \beta$ with $a = [1.5, 0.5]^T$ and $\beta = 0$; (b) the nonlinearity $\sigma(\langle a, x\rangle - \beta)$ with the same values for a and β; and (c) the nonlinearity $\sigma(\langle a', x\rangle - \beta)$ with $a' = 4a$ and $\beta = 12$. Show are the output values for points in a rectangular region of the feature space (top row), and for points x with $[x]_2 = 0$ (bottom row).

2.1 Definition of halfspaces

The output of the first layer in the network can be written as $y = \sigma(Ax - b)$ with $x \in \mathbb{R}^d$. For an individual unit j this reduces to $\sigma(\langle a, x\rangle - \beta)$, where $a \in \mathbb{R}^d$ corresponds to $[A]_j$, the j-th row of A, and $\beta = [b]_j$. When applied over all points of the feature space, the affine mapping $\langle a, x\rangle - \beta$ generates a linear gradient, as shown in Figure 2(a). The output of unit is then obtained by applying the sigmoid function to these intermediate values. When doing so, assuming throughout that $a \neq 0$, two prominent regions form: one with values close to -1 and one with values close to $+1$. In between the two regions there is a smooth transition region, as illustrated in Figure 2(b). The center of the transition region consists of all feature points whose output value equal zero. It can be verified that this set is given by all points $x = \beta a/\|a\|_2^2 + v$ such that $\langle a, v\rangle = 0$, and therefore describes a hyperplane. The normal direction of the hyperplane is given by a, and the exact location of the hyperplane is determined by a shift along this normal, controlled by both β and $\|a\|_2$. The region of all points that map to nonnegative values forms a halfspace, and because the linear functions can be chosen independently for each unit, we can define as many halfspaces are there are units in the first layer. As the transition between the regions on either side of the hyperplane is gradual it is convenient to work with soft boundaries and interpret the output values as a confidence level of set membership with values close to $+1$ indicating strong membership, those close to -1 indicating strong non-membership, and with decreasing confidence levels in between. For simplicity we use the term halfspace for both the soft and sharp versions of the region.

In addition to normal direction and location, halfspaces are characterized by the sharpness of the transition region. This property can be controlled in two similar ways (see also Section 5.2). The first is to scale both a and β by a positive scalar γ. Doing so does not affect the location or orientation of

the hyperplane but does scale the input to the sigmoid function by the same quantity. As a consequence, choosing $\gamma > 1$ shrinks the transition region, whereas choosing $\gamma < 1$ causes it to widen. The second way is to replace the activation function σ by σ_γ. Scaling only a affects the sharpness of the transition in the same way, but also results in a shift of the hyperplane along the normal direction whenever $\beta \neq 0$. Note however that the activation functions are typically fixed and the properties of the halfspaces are therefore controlled only by the weight and bias terms. Figure 2(c) illustrates the sharpening of the halfspace and the use of β to change its location.

2.2 Combination of intermediate regions

The second layer combines the halfspace regions defined in the first layer resulting in new regions in each of the output nodes. In case step-function activation functions are used, the operations used to combine the regions correspond to set operations including complements (^{c}), intersection ($\cap$), and unions ($\cup$). The same operations are used in subsequent layers, thereby enabling the formation of increasingly complex regions. The use of a sigmoidal function instead of the step function does not significantly change the types of operations, although some care needs to be taken.

Some operations are best explained when working with input coming from the logistic function (with values ranging from 0 to 1) rather than from the sigmoid function (ranging from -1 to 1). Note however that output x from a sigmoid can easily be mapped to the equivalent output $x' = (x-1)/2$ from a logistic function, and vice versa. Any linear operation $Ax' - b$ on the logistic output then becomes $A(x-1)/2 - b = \tilde{A}x - \tilde{b}$ with $\tilde{A} = A/2$ and $\tilde{b} = b + Ae/2$. In other words, with appropriate changes in A and b we can always choose which of the two activation functions the input comes from, regardless of which function was actually used.

2.2.1 Elementary Boolean operations

To make the operations discussed in this section more concrete we apply them to input generated by a first layer with the following parameters:

$$A_1 = \begin{bmatrix} 9 & 1 \\ -2 & 6 \end{bmatrix}, \quad b_1 = \begin{bmatrix} -2 \\ -1 \end{bmatrix}, \quad \text{and} \quad \nu_1 = \sigma_3.$$

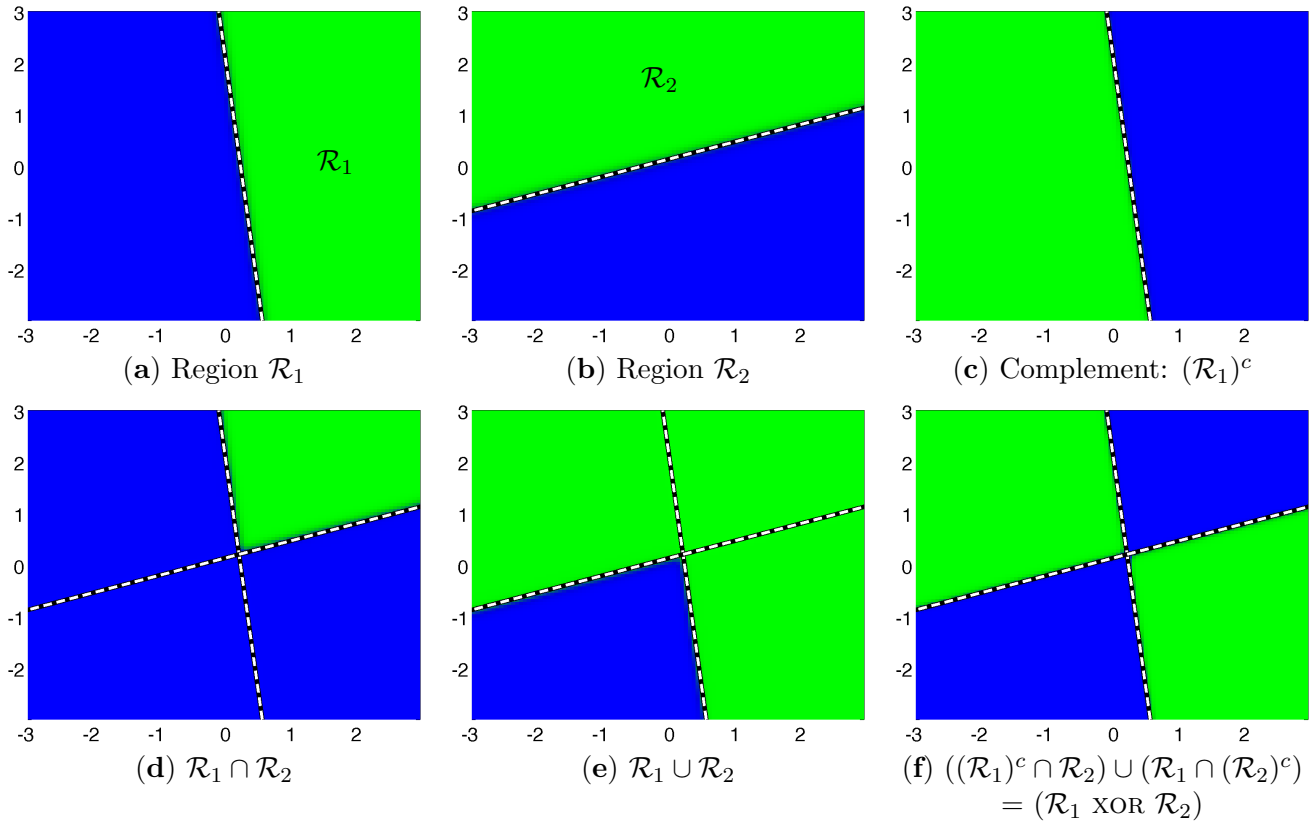
The two resulting halfspace regions $\mathcal{R}_1$ and $\mathcal{R}_2$ are illustrated in Figures 3(a) and 3(b). For simplicity we denote the parameters for the second layer by A and b, omitting the subscripts. In addition, we omit all entries that are not relevant to the operation and appropriate padding with zeros where needed is implied.

Constants Constants can be generated by choosing $A = 0$ and choosing a sufficiently large positive or negative offset values. For example, choosing $b = 100$ gives a region that spans the entire domain (representing the logical TRUE), whereas choosing $b = -100$ results in the empty set (or logical FALSE).

Unary operations The simplest unary operation, the identity function, can be defined as

$$A_I = 1, \qquad b_I = 0. \tag{Identity}$$

This function works well when used in conjunction with a step function, but has an undesirable damping effect when used with the sigmoid function: input values up to 1 are mapped to output values up to $\sigma(1) \approx 0.46$, and likewise for negative values. While such scaling may be desirable in certain cases, we would like to preserve the clear distinction between high and low confidence regions. We can do this by scaling up A, which amplifies the input to the sigmoid function and therefore its output. Choosing



(**a**) Region $\mathcal{R}_1$ (**b**) Region $\mathcal{R}_2$ (**c**) Complement: $(\mathcal{R}_1)^c$

(**d**) $\mathcal{R}_1 \cap \mathcal{R}_2$ (**e**) $\mathcal{R}_1 \cup \mathcal{R}_2$ (**f**) $((\mathcal{R}_1)^c \cap \mathcal{R}_2) \cup (\mathcal{R}_1 \cap (\mathcal{R}_2)^c)$ $= (\mathcal{R}_1 \text{ XOR } \mathcal{R}_2)$

Figure 3: Illustration of (a),(b) regions defined by the first neural network layer, and (c)–(f) various Boolean set operations applied to them in subsequent layers.

$A_I = 3$, for example, would increase the maximum confidence level to $\sigma(3) \approx 0.91$. As noted towards the end of Section 2.1, the same can be achieved by working with σ_3, and to avoid getting distracted by scaling issues like these we will work with $\nu_2 = \sigma_3$ throughout this section. We note that the identity function can be approximated very well by scaling down the input and taking advantage of the near-linear part of the sigmoid function around zero. The output can then be scaled up again in the next layer to achieve the desired result. Similar to the identity function, we define the complement of a set as

$$A_c = -1, \qquad b_c = 0. \tag{Complement}$$

The application of this operator to $\mathcal{R}_1$ is illustrated in Figure 3(c). Just to be clear, note that in this case the full parameters to the second layer would be $A = [-1, 0]$ and $b = 0$.

Binary operations When taking the intersection of regions $\mathcal{R}_1$ and $\mathcal{R}_2$ we require that the output values of the corresponding units in the network sum up to a value close to two. This is equivalent to saying that when we subtract a relatively high value, say 1.5, from the sum, the outcome should remain positive. This suggests the following parameters for binary intersection:

$$A_\cap = [1, 1], \qquad b_\cap = 1.5. \tag{Intersection}$$

We now combine the intersection and complement operations to derive the union of two sets, and to illustrate how complements of sets can be applied during computations. By De Morgan's law, the

union operator can be written as $\mathcal{R}_1 \cup \mathcal{R}_2 = ((\mathcal{R}_1)^c \cap (\mathcal{R}_2)^c)^c$. Evaluation of this expression is done in three steps: taking the individual complements of $\mathcal{R}_1$ and $\mathcal{R}_2$, applying the intersection, and taking the complement of the result. This can be written in linear form as

$$A_c \left(A_\cap \left(\begin{bmatrix} A_I & \\ & A_c \end{bmatrix} x + \begin{bmatrix} b_I \\ b_c \end{bmatrix} \right) + b_\cap \right) + b_c.$$

Substituting the weight and bias terms and simplifying yields parameters for the union:

$$A_\cup = [1, 1], \qquad b_\cup = -1.5. \tag{Union}$$

It can be verified that the intersection can similarly be derived from the union operator based using $\mathcal{R}_1 \cap \mathcal{R}_2 = ((\mathcal{R}_1)^c \cup (\mathcal{R}_2)^c)^c$ Results obtained with both operators are shown in Figures 3(e) and 3(f).

2.2.2 General n-ary operations

We now consider general operations that combine regions from more than two units. It suffices to look at a single output unit $\sigma(\langle a, x\rangle - \beta)$ with weight vector a and bias term β. Any negative entry in a mean that the corresponding input region is negated and that its complement is used, whereas zero valued entries indicate that the corresponding region is not used. Without loss of generality we assume that all input regions are used and normalized such that all entries in a can be taken strictly positive. We again start by looking at the idealized situation where inputs are generated using a step function with outputs -1 or 1. When a is the vector of all ones, and k out of n inputs are positive we have $\langle a, x\rangle = k - (n - k) = 2k - n$. Choosing activation level $\beta = 2k - n - 1$ therefore ensures that the output of the unit is positive whenever at least k out of n inputs are positive. As extreme cases of this we obtain the n-ary intersection with $k = n$, and the n-ary union by choosing $k = 1$. Weights can be adjusted to indicate how many times each region gets counted.

It was noted by Huang and Littmann [12] that complicated and highly non-intuitive regions can be formed with the general n-ary operations, even in the second layer. As an example, consider the eight hyperplanes plotted in Figure 4(a). The weight assigned to each hyperplane determines the contribution to each cell that lies within the enclosed halfspace. The total contributions for each cell shown in Figure 4(a) represent the total weight obtained when using weight two to the outer hyperplanes and a unit weight for the inner hyperplanes, combined with step function input ranging from 0 to 1. Adding up values for so many regions in a single step worsens the scaling issue mentioned for the unitary operator: In this case choosing a threshold of $\beta = 9.5$ leads to values ranging from -3.5 to 0.5 before application of the sigmoid function. Using $\nu_1 = \ell_{10}$ and $\nu_2 = \sigma_{50}$ for amplification with different weight vectors and threshold values we obtain the regions shown in Figures 4(b) to 4(d).

Removing the large amplification factor in the second layer can lead to regions with low or varying confidence levels. For the mixed weights example, using $\nu_2 = \sigma_1$ and threshold $\beta = 6.5$ causes the intended region to have four distinct confidence levels, as shown in Figure 4(e). Low weights can also be leveraged to obtain a parsimonious representation of smooth regions that would otherwise require the many more halfspaces. An example of this is shown in Figure 4(f) in which the four outer halfspaces with soft boundaries are combined to form a smooth circular region.

2.3 Boolean function representation using two layers

As seen from Section 2.2.1 neural networks can be used to take the union of intersections of (possibly negated) sets. In Boolean logic this form is called disjunctive normal form (DNF), and it is well known that any Boolean function can be expressed in this form (see also [1]). Likewise we could reverse the order

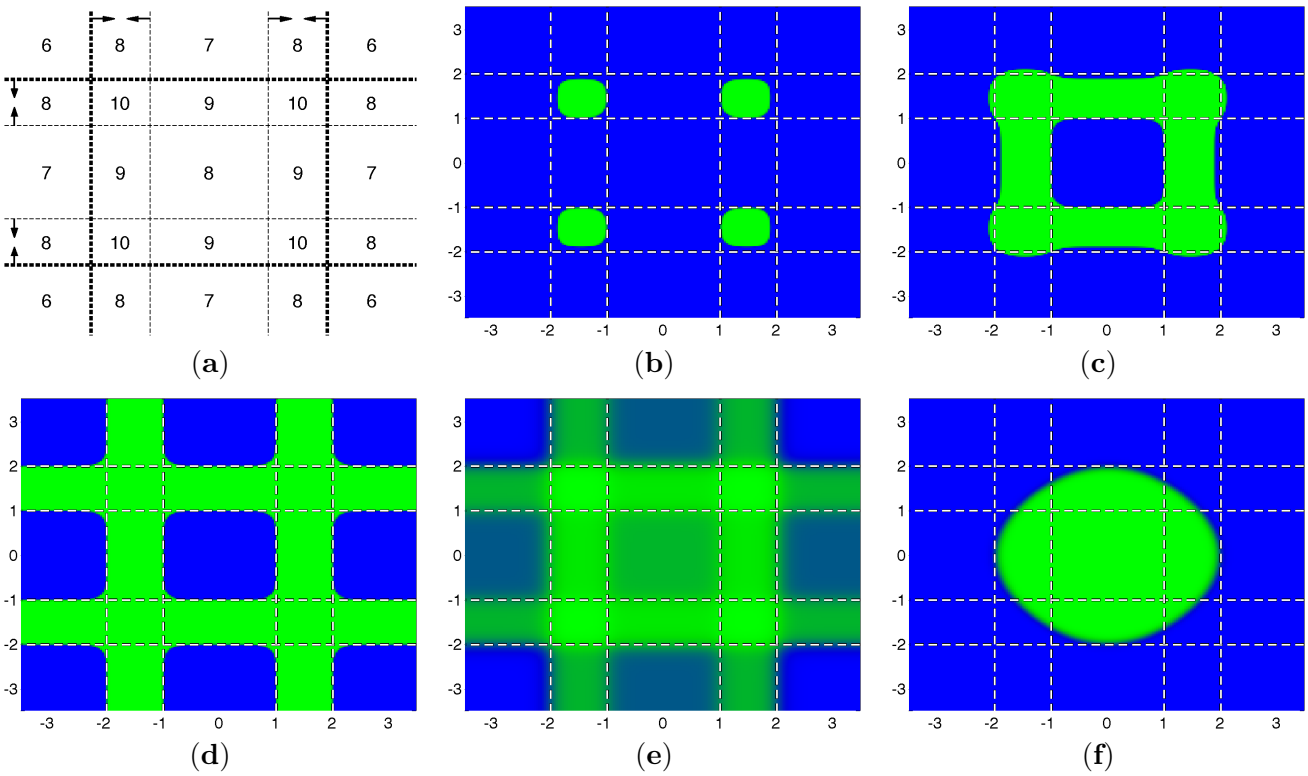


Figure 4: Application of the n-ary operator. Plot (a) shows the location and orientation of the eight hyperplanes and the total weight for each of the cells when using weight two for the outer hyperplanes (thick dashed line) and unit weight for the inner hyperplanes (thin dashed line). Plots (b) and (c) show the regions formed when choosing $\beta = 9.5$ and $\beta = 8.5$ respectively, with activation function $\nu_1 = \ell_{10}$ and $\nu_2 = \sigma_{50}$. Plot (d) shows the region obtained when assigning unit weights to each hyperplanes and using $\beta = 4.5$. In plot (e) we change the settings from plot (b) by replacing the second activation function to σ_1 and using $\beta = 6.5$. The lack of amplification results in a region with four different confidence levels. Plot (f) illustrates the formation of a smooth circular region using only the outer four hyperplanes together with activation functions $\nu_1 = \ell_1$ and $\nu_2 = \sigma_{50}$, and threshold $\beta = 6.5$.

of the union and intersection operators and arrive at conjunctive normal form (CNF), which is equally powerful. Two-layer networks are, in fact, far stronger than this and can be used to approximate general smooth functions. More information on this can be found in [2, Sec. 4.3.2].

2.4 Boundary regions and amplification

The use of sigmoidal nonlinearity functions leads to continuous transitions between different regions. The center of the transition regions for a node can be defined as the set of feature points for which the output of that node is zero. Given input x_{k-1} for some node at depth k we first form $\langle a, x_{k-1} \rangle$, then subtract the bias β, and apply the sigmoid function. The output is zero if only if $\langle a, x_{k-1} \rangle = \beta$, and the transition center therefore corresponds to the level set of $\langle a, x_{k-1} \rangle$ at β. For a fixed a we can thus control the location of the transition by changing β. As an example consider a two-level neural network with the first layer parameterized by $A_1 = I$, $b_1 = 0$, and the second layer by $A_2 = [1, 1]$, $b = \beta$. Writing the input vector as $x_0 = [x, y]$ it can be seen that $A_2 x_1 = \sigma(x) + \sigma(y)$, as illustrated in Figure 5. All values greater than β will be mapped to positive values and, as discussed in Section 2.2.1, we again see that

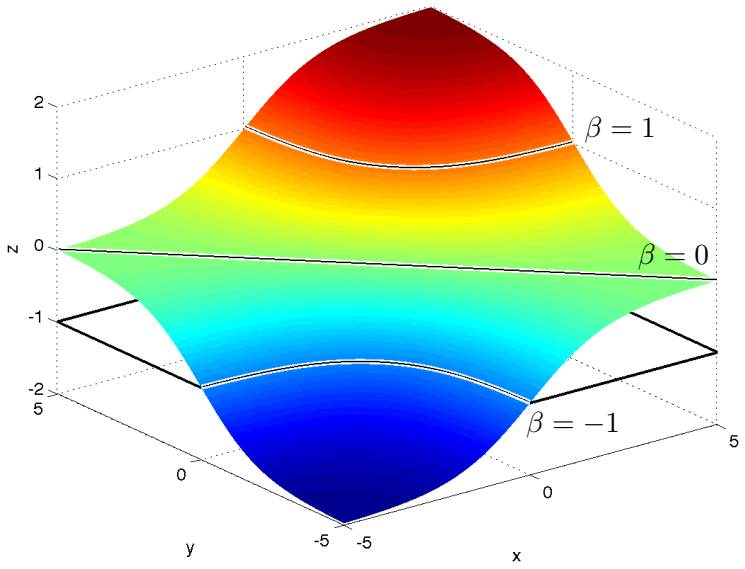


Figure 5: Level sets of $z = \sigma(x) + \sigma(y)$ at values $\beta = -1, 0, 1$ along with a slice at $\beta = -1$.

choosing $\beta > 0$ approximates the intersection of the regions $x \geq 0$ and $y \geq 0$, whereas choosing $\beta < 0$ approximates the union (indicated in the figure by the lines at $z = -1$). What we are interested in here is the location of the transition center. Clearly, making the intersection more stringent by increasing β causes the boundary to shift and the resulting region to become smaller. Another side effect is that the output range of the second layer, which is given by $[\sigma(-2-\beta), \sigma(2-\beta)]$, changes. Choosing β close to 2, the supremum of the input signal, means that the supremum of the output is close to zero, whereas the infimum nearly reaches -1. To obtain larger positive confidence levels in the output, without shifting the transition center, we need to amplify the input by scaling A_2 and b_2 by some $\gamma > 1$. In Figure 6 we study several aspects of the boundary region corresponding to the setting used for Figure 5, with the addition of scaling parameter γ. For a given β we choose γ such that the maximum output of $\sigma(\gamma(2-\beta))$ is 0.995. Figures 6(a)–(c) show the transition region with values ranging from -0.95 and 0.95 along with the center of the transition with value 0 and the region with values exceeding 0.95. Figure 6(d) shows the required scaling factors.

The ideal intersection of the two regions coincides with the positive orthant and we define the shift in the transition boundary as the limit of the y-coordinate of the zero crossing as x goes to infinity, giving

$$\lim_{x \to \infty} \sigma^{-1}(\beta - \sigma(x)) = \sigma^{-1}(\beta - 1).$$

The resulting shift values are show in Figure 6(e). Another property of interest is the width of the transition region. Similar to the shift we quantify this as the difference between the asymptotic y-coordinates of the -0.95 and 0.95 level set contours as x goes to infinity. We plot the results for several multiples of γ in Figure 6(f). As expected, we can see that larger amplification reduces the size of the transition intervals. The vertical dashed line indicates the critical value of β at which the -0.95 contour becomes diagonal ($y = -x$) causing the transition width to become infinite. The same phenomenon happens at smaller β when the multiplication factor is higher. Note that this break down is due only to the definition of the transition width; the transition region itself remains perfectly well defined throughout.

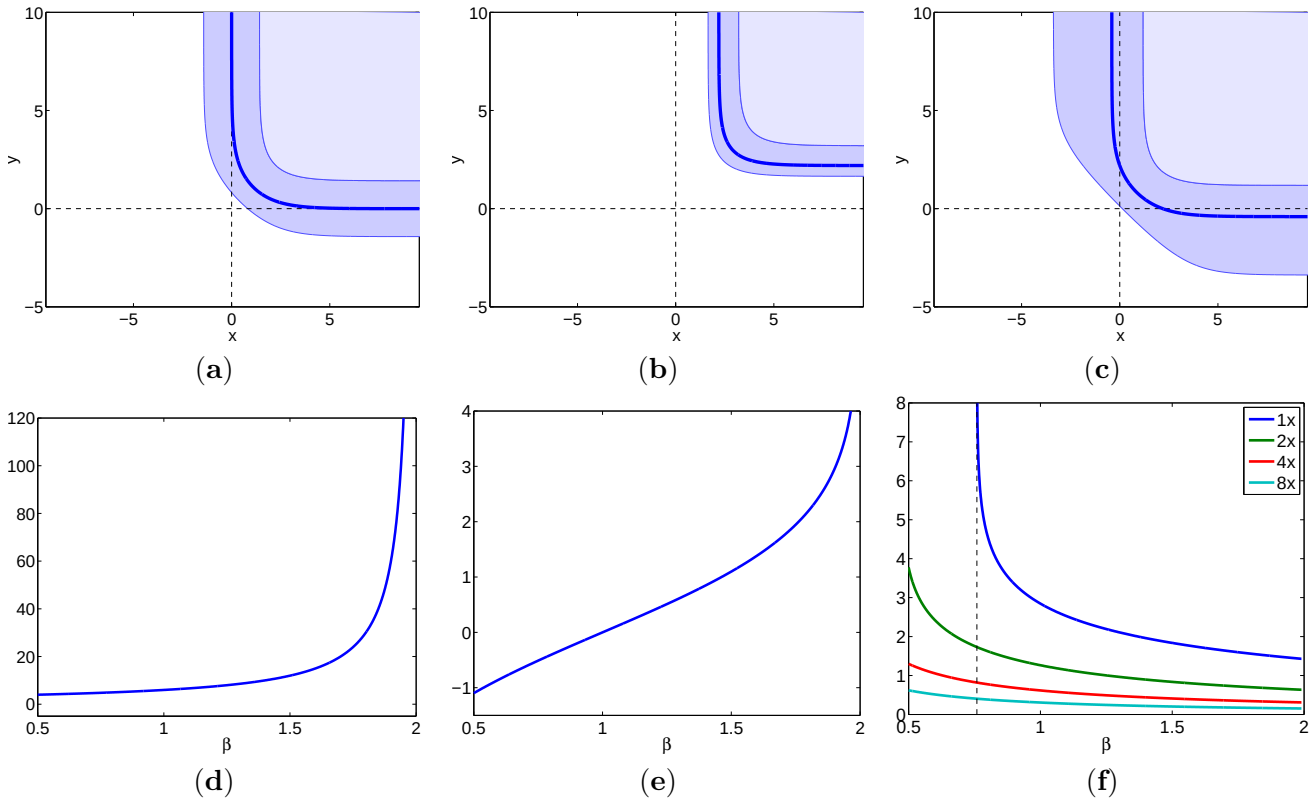


Figure 6: Transition regions for $\sigma_\gamma(\sigma(x)+\sigma(y)-\beta)$ with contour lines at 0 and 0.95 of the minimum and maximum values for (a) $\beta = 1$, (b) $\beta = 1.8$, and (c) $\beta = 0.6$. The value of scaling factor γ is chosen such that output range reaches at least ± 0.995. Plot (d) shows the required scaling factor as a function of β. The shift in the transition region is plotted in (e). Plot (f) shows the transition width as a function of β, using different multiples of γ.

2.5 Continuous to discrete

The level-set nature of applying the nonlinearity as illustrated in Figure 5 allows the generation of decision boundaries that look very different from any one of those used for its input. One example of this was shown in Figure 4(f) in which a circular region was generated by four axis-aligned hyperplanes, and we now describe another. Consider the two hyperplanes in Figures 7(a,b), generated in the first layer with respectively $a = [0.1, -0.1]$, $b = 0$, and $a = [0.1, 0.1]$, $b = 0$. The small weights and the limited domain size cause the input values to the sigmoid nonlinearity to be small. As a result, the sigmoid operates in its near-linear region around the origin and therefore resembles scalar multiplication. Consequently, because the normals of the first layers form a basis, we can use the second layer to approximate any operation that would normally occur in the first layer. For example we can choose $a_2 = [\cos(\alpha + \pi/4), \sin(\alpha + \pi/4)]$ and $b_2 = 0$ to generate a close approximation of a hyperplane at angle α (up to a scaling factor this weight matrix is formed by multiplying the desired normal vector a by the rotation on the inverse of the weight matrix of the first layer). The resulting regions of the second layer are shown in Figures 7(c,d) for $\alpha = 90°$ and $\alpha = 70°$, respectively. This illustrates that, although somewhat contrived, it is technically possible to change hyperplane orientation after the first layer.

As decision regions propagate and form through one or more layers with modest or large weights, their boundaries become sharper and we see a gradual transition from continuous to discrete network behavior.

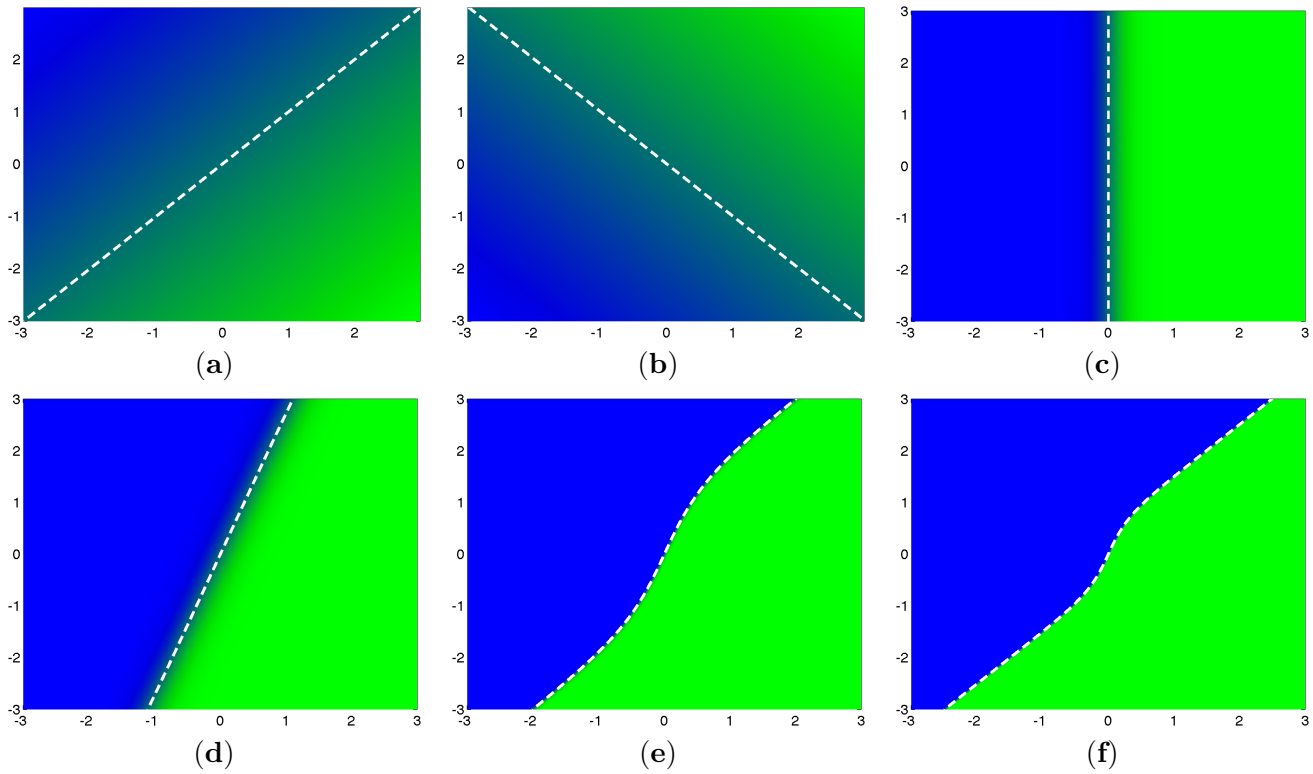


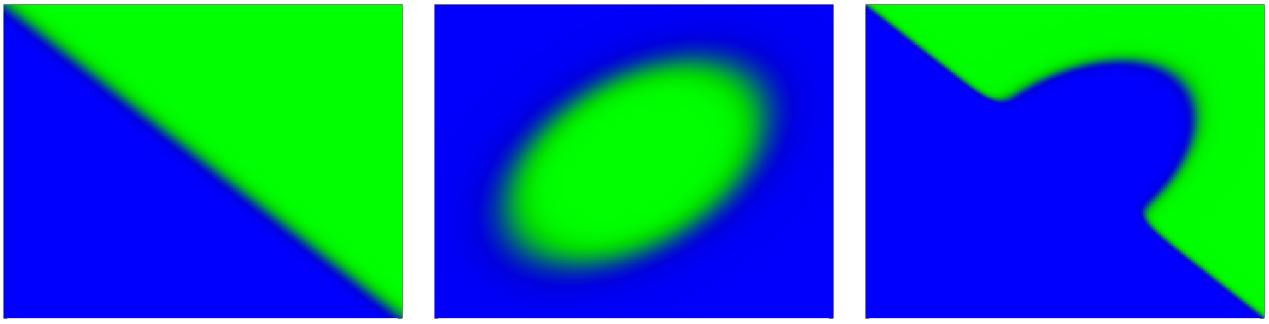
Figure 7: Combination of (a,b) two smooth regions defined by a diagonal hyperplane into (c) a vertical region, and (d) a region at an angle of 70 degrees. Using the setup for (d) with scaled weights in the first layer gives the region shown in (e) for weight factor 10, and (f) for weight factor 20.

In the continuous regime, where the transitions are still gradual, the decision boundaries emerge as level sets of slowly varying smooth functions and therefore change continuously and considerably with the choice of bias term. As the boundary regions become sharper the functions tend to piecewise constant causing the level sets to change abruptly only at several critical values while remaining fairly constant otherwise, thus giving more discrete behavior. In Figures 7(e,f) we show intermediate stages in which we scale the weights in the first layer Figures 7(d) by a factor of 10 and 20, respectively. In addition, it can be seen that scaling in this case does not just sharpen the boundaries, but actually severely distorts them. Finally, it can be seen that the resulting region becomes increasingly diagonal (similar to its sharpened input) as the weights increase. This again emphasizes the more discrete nature of region combinations once the boundaries of the underlying regions are sharp.

2.6 Generalized functions for the first layer

The nodes in the first layer define geometric primitives, which are combined in subsequent layers. Depending on the domain it may be desirable to work with primitives other than halfspaces, or to provide a set of different types. This can be achieved by replacing the inner products in the first layer by more general functions $f_\theta(x)$ with training examples x and (possibly shared) parameters θ. The traditional hyperplane is given by

$$f_\theta(x) = \langle a, x \rangle + \beta, \qquad \theta = (a, \beta)$$



(**a**) Linear, $\mathcal{R}_1$ (**b**) Gaussian, $\mathcal{R}_2$ (**c**) $\mathcal{R}_1 \cap \mathcal{R}_2^c$

Figure 8: Shape primitives of type (a) halfspace, and (b) Gaussian, are combined to obtain (c).

For ellipsoidal regions we could then use

$$f_\theta(x) = \alpha \|Ax - b\|_2^2 + \beta, \qquad \theta = (A, b, \alpha, \beta).$$

More generally, it is possible to redefine the entire unit by replacing both the inner-product and the nonlinearity with a general function to obtain, for example, a radial-basis function unit [14]. In Figure 8 we illustrate how a mixture of two types of geometric primitives can form regions that cannot be expressed concisely with either type alone.

3 Region properties and approximation

The hyperplanes defined by the first layer of the neural network partition the space into different regions. In this section we discuss several combinatorial and approximation theoretic properties of these regions.

3.1 Number of regions

One of the most fundamental properties to consider is the maximum number of regions into which $\mathbb{R}^d$ can be partitioned using n hyperplanes. The exact maximum is well known to be

$$r(n, d) = \sum_{i=0}^{d} \binom{n}{i}, \tag{6}$$

and is attained whenever the hyperplanes are in general position [20, p.39]. With the hyperplanes in place, the subsequent logic layers in the neural network can be used to identify each of these regions by taking the union of (complements of) halfspaces. Individual regions can then be combined using the union operator.

3.2 Approximate representation of classes

3.2.1 Polytope representation

When the set of points $\mathcal{C} \subset \mathbb{R}^d$ belonging to a class form a bounded convex set, we can approximate it by a polytope $\mathcal{P}$ given by the bounded intersection of a finite number of halfspaces. The accuracy of such an approximation can be expressed as the Hausdorff distance between the two sets, defined as:

$$\rho_H(C, \mathcal{P}) := \max \left[\sup_{x \in \mathcal{C}} d(x, \mathcal{P}),\ \sup_{x \in \mathcal{P}} d(x, \mathcal{C}) \right],$$

with

$$d(x, \mathcal{S}) := \inf_{y \in \mathcal{S}} \|x - y\|_2.$$

For a given class of convex bodies Σ, denote $\delta_H(\mathcal{C}, \Sigma) := \inf_{\mathcal{V} \in \Sigma} \rho_H(\mathcal{C}, \mathcal{V})$. We are interested in $\delta_H(\mathcal{C}, \Sigma)$ when $\Sigma = \mathfrak{R}^d_{(n)}$, the set of all polytopes in $\mathbb{R}^d$ with at most n facets (i.e., generated by the intersection of up to n halfspaces), and in particular how it behaves as a function of n. The following result obtained independently by [4, 7] is given in [5]. For every convex body $\mathcal{U}$ there exists a constant $c(\mathcal{U})$ such that

$$\delta_H(\mathcal{U}, \mathfrak{R}^d_{(n)}) \leq \frac{c(\mathcal{U})}{n^{2/(d-1)}}.$$

More interesting perhaps is a lower bound on the approximation distance. For the unit ball $\mathcal{B}$ we have the following:

Theorem 3.1. *Let $\mathcal{B}$ denote the unit ball in $\mathbb{R}^d$. Then for sufficiently large n there exists a constant c_d such that*

$$\delta_H(\mathcal{B}, \mathfrak{R}^d_{(n)}) \geq \frac{c_d}{n^{2/(d-1)}},$$

Proof. For n large enough there exists a polytope $\mathcal{P} \in \mathfrak{R}^d_{(n)}$ with n facets and $\delta := \delta_H(\mathcal{B}, \mathcal{P}) \leq 9/64$. Each of the n facets in $\mathcal{P}$ is generated by a halfspace, and we can use each halfspace to generate a point on the unit sphere in $\mathbb{R}^d$ such that the surface normal at that point matches the outward normal of the halfspace. We denote the set of these points by $\mathcal{N}$, with $|\mathcal{N}| = n$. Now, take any point x on the unit sphere. From the definition of δ it follows that the maximum distance between x and the closest point on one of the hyperplanes bounding the halfspaces is no greater than δ. From this it can be shown that the distance to the nearest point in $\mathcal{N}$ is no greater than $\epsilon := 2\sqrt{\delta}$. Moreover, because the choice of x was arbitrary, it follows that $\mathcal{N}$ defines an ϵ-net of the unit sphere. Lemma 3.2 below shows that the cardinality $|\mathcal{N}| \geq \frac{c'}{\epsilon^{(d-1)}}$. Substituting $\epsilon = 2\sqrt{\delta}$ gives

$$n \geq \frac{c'}{2^{(d-1)}\delta^{(d-1)/2}}, \quad \text{or} \quad \delta \geq \frac{c_d}{n^{2/(d-1)}}.$$

□

Lemma 3.2. *Let $\mathcal{N}$ be an ϵ-net of the unit sphere $\mathcal{S}^{d-1}$ in $\mathbb{R}^d$ with $\epsilon \leq 3/4$, then*

$$|\mathcal{N}| \geq \sqrt{2\pi(d-1)/d} \cdot \epsilon^{1-d}.$$

Proof. By definition of the ϵ-net, we obtain a cover for $\mathcal{S}^{d-1}$ by placing balls of radius ϵ at all $x \in \mathcal{N}$. The intersection of each ball with the sphere gives a spherical cap. The union of the spherical caps covers the sphere and $|\mathcal{N}|$ times the area of each spherical cap must therefore be at least as large as the area of the sphere. A lower bound on the number of points in $\mathcal{N}$ is therefore obtained by the ratio ν between the area of the sphere and that of the spherical cap (see also [23, Lemma 2]). Denoting by $\varphi = \arccos(1 - \frac{1}{2}\epsilon^2)$ the half-angle of the spherical cap it follows from [13, Corollary 3.2(iii)] that ν satisfies

$$1/\nu < \frac{1}{\sqrt{2\pi(d-1)}} \cdot \frac{1}{\cos\varphi} \cdot \sin^{d-1}\varphi,$$

whenever $\varphi \leq \arccos 1/\sqrt{d}$. This bound can be substituted into the second term above to obtain $\sqrt{d}$, and can be verified to hold whenever $\epsilon \leq 3/4$. It further holds that $\sin\varphi < \epsilon$ which, after rewriting, gives the desired result. □

3.2.2 More efficient representations

From Theorem 3.1 we see that a large number of supporting hyperplanes is needed to define a polytope that closely approximates the unit ℓ_2-norm ball. Approximating such a ball or any other convex sets by the intersection of a number of halfspaces can be considered wasteful, however, since it uses only a single region out of the maximum $r(n, d)$ given by (6). This fact was recognized by Cheang and Barron [6], and they proposed an alternative representation for unit balls that only requires $\mathcal{O}(d^2/\delta^2)$ halfspaces—far fewer than the conventional $\mathcal{O}(1/\delta^{(d-1)/2})$. The construction is as follows: given a set of n suitably chosen halfspaces $\mathcal{H}_i$ and the indicator function $1_{\mathcal{H}_i}(x)$ which is one if $x \in \mathcal{H}_i$ and zero otherwise. Typically these halfspaces are used to define polytope $\mathcal{P} := \{x \in \mathbb{R}^d \mid \sum_i 1_{\mathcal{H}_i}(x) = n\}$, that is, the intersection of all halfspaces. The (non-convex) approximation proposed in [6] is of the form

$$\mathcal{Q} := \{x \in \mathbb{R}^d \mid \sum_i 1_{\mathcal{H}_i}(x) \geq k\},$$

which consists of all points that are contained in at least k halfspaces. This representation is shown to provide far more efficient approximations, especially in high dimensions. As described in Section 2.2.2, this construction can easily be implemented as a neural network. A similar approximation for the Euclidean ball, which also takes advantage of smooth transition boundaries is shown in Figure 4(f).

3.3 Bounds on the number of separating hyperplanes

In many cases, it suffices to simply distinguish between the different classes instead of trying to exactly trace out their boundaries. Doing so may reduce the number of parameters and additionally help reduce overfitting. The bound in Section 3.1 gives the maximum number of regions that can be separated by a given number of hyperplanes. Classes found in practical applications are extremely unlikely to exactly fit these cells, and we can therefore expect that more hyperplanes are needed to separate them. We now look at the maximum number of hyperplanes that is needed.

3.3.1 Convex sets

In this section we assume that the classes are defined by convex sets whose intersection is either empty or of measure zero. We are interested in finding the minimum number of hyperplanes needed such that each pair of classes is separated by at least one of the hyperplanes. In the worst case, a hyperplane is needed between any pair of n classes, giving a maximum of $\binom{n}{2}$ hyperplanes, independent of the ambient dimension. That this maximum can be reached was shown by Tverberg [22] who provides a construction due to K.P. Villanger of a set of n lines in $\mathbb{R}^3$ such that any hyperplane that separates one pair of lines, intersects all others. Here we describe a generalization of this construction for odd dimensions $d \geq 3$.

Theorem 3.3. *Let $A = [A_1, A_2, \ldots, A_n]$ be a full-spark matrix with blocks A_i of size $d \times (d-1)/2$, with odd $d \geq 3$. Let b_i, $i = 1, \ldots, n$ be vectors in $\mathbb{R}^d$ such that $[A_i, A_j, b_i - b_j]$ is full rank for all $i \neq j$. The subspaces*

$$\mathcal{S}_i = \{x \in \mathbb{R}^d \mid x = A_i v + b_i,\ v \in \mathbb{R}^{(d-1)/2}\}.$$

are pairwise disjoint and any hyperplane separating $\mathcal{S}_i$ and $\mathcal{S}_j$, $i \neq j$, intersects all $\mathcal{S}_k$, $k \neq i, j$.

Proof. Any pair of subspaces $\mathcal{S}_i$ and $\mathcal{S}_j$ intersects only if there exist vectors u, v such that

$$A_i u + b_i = A_j v + b_j, \quad \text{or} \quad [A_i, A_j] \begin{bmatrix} u \\ -v \end{bmatrix} = b_j - b_i.$$

It follows from the assumption that $[A_i, A_j, b_j - b_i]$ is full rank, that no such two vectors exist, and therefore that all subspaces are pairwise disjoint.

Any hyperplane $\mathcal{H}_{i,j}$ separating $\mathcal{S}_i$ and $\mathcal{S}_j$ is of the form $a^T x = \beta$. To avoid intersection with $\mathcal{S}_i$ we must have $a^T(A_i v + b) \neq \beta$ for all $v \in \mathbb{R}^{d-1}$, which is satisfied if an only if $a^T A_i = 0$. It follows that we must also have $a^T A_j = 0$, and therefore that a is a normal vector to the $(d-1)$-subspace spanned by $[A_i, A_j]$. From the full-spark assumption on A it follows that $a^T A_k \neq 0$ for all $k \neq i, j$, which shows that $\mathcal{H}_{i,j}$ intersects the corresponding $\mathcal{S}_k$. The result follows since the choice of i and j was arbitrary. □

Random matrices A and vectors b_i with entries i.i.d. Gaussian satisfy the conditions in Theorem 3.3 with probability one, thereby showing the existence of the desired configurations. A simple extension of the construction to dimension $d+1$ is obtained when generating subspaces $\mathcal{S}'_i \subset \mathbb{R}^{d+1}$ by matrices A'_i, formed by appending a row of zeros to A_i and adding a column corresponding to the last column of the $d \times d$ identify matrix, and vectors $b'_i = [b_i; 0]$. Pach and Tardos [17] further show that the lines in the construction described by Tverberg can be replaced by appropriately chosen unit segments. Adding a sufficiently small ball in the Minkowski sense then results in n bounded convex sets with non-empty interior whose separation requires the maximum $\binom{n}{2}$ hyperplanes.

3.3.2 Point sets

When separating a set of n points, the maximum number of hyperplanes needed is easily seen to be $n-1$; we can cut off a single extremal point of subsequent convex hulls until only a single point is left. This maximum can be reached, for example when all points lie on a straight line. For a set of points in general position, it is shown in [3] that the maximum number $f(n,d)$ of hyperplanes needed satisfies

$$\lceil (n-1)/d \rceil \leq f(n,d) \leq \lceil (n - 2^{\lceil \log d \rceil})/d \rceil + \lceil \log d \rceil.$$

Based on this we can expect the number of hyperplanes needed to separate a family of unit balls to be much smaller than the maximum possible $\binom{n}{2}$, whenever $n > d+1$.

3.3.3 Non-convex sets

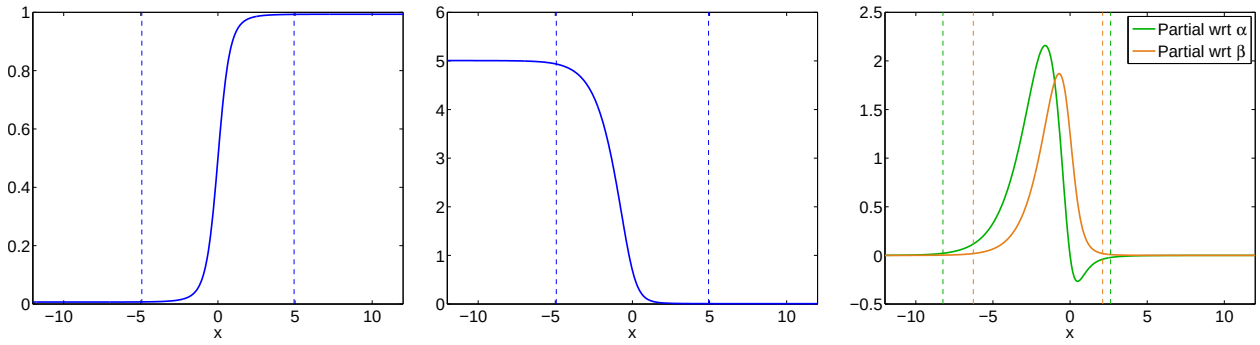
The interface between two non-convex sets can be arbitrarily complex, which means that there are no meaningful bounds on the number of hyperplanes needed to separate general sets.

4 Gradients

Parameters in the neural network are learned by minimizing a loss function over the training set, using for example backpropagation [19] or stochastic gradient descent on the formulation shown in (4). The gradient of such a loss function decouples over the training samples and can be written as

$$\nabla \phi(s) = \tfrac{1}{|\mathcal{T}|} \sum_{(x,c) \in \mathcal{T}} \nabla f(s; x, c) \tag{7}$$

The idea of the section is to explore how points contribute when they are part of a training set. That is, for a given parameter set s, and with the class information $c = c(x)$ assumed to be known, we are interested in $\nabla f(x)$, behavior of ∇f as a function of x. We will see that some points in the training set contribute more to the gradient than others. So, instead of just looking at the total gradient, we can also look at the amount of information that is provided by each point: points that have a large relative contribution to the gradient can be said to be more informative than those that do not contribute much.



(**a**) (**b**) (**c**)

Figure 9: Classification of points on the x-axis, with (a) the decision region: $\hat{p}(x) = \ell_5(\sigma(\alpha x - \beta))$ with $\alpha = 1$, $\beta = 0$; (b) the cross entropy: $f(x) = -\log \hat{p}(x)$; (c) partial derivatives of the loss function with respect to α and β

Throughout this and the next section we use the word 'gradient' loosely and also use it to refer to blocks of gradient entries corresponding to the parameters a layer, individual entries, or the gradient field $\nabla f(x)$ of those quantities over the entire feature space. The exact meaning should be clear from the context.

4.1 Motivational example

We illustrate the relative importance of different training samples using a simple example. We define a basic two-layer neural network in which the first layer defines a hyperplane $\alpha x = \beta$ with nonlinearity $\nu_1 = \sigma$, and in which the second layer applies the identity function follow by nonlinearity $\nu_2 = \ell_5$ for amplification. Choosing $\alpha = 1$ and $\beta = 0$ defines the region shown in Figure 9(a). Now, suppose that all points $x \in [-12, 12]$ belong to the same class and should therefore be part of this region. Intuitively, it can be seen that slight changes in the location of the hyperplane or in the steepness of the transition will have very little effect on the output of the neural network for input points $|x| \geq 5$, say, since values close to one or zero remain so after the perturbation. As such, we expect that in these regions the gradient with respect to α and β will be small. For points in the transition region the change will be relatively large, and the gradient at those points will therefore be larger. This suggests that training point away from the transition region provide little information when deciding in which direction to move the hyperplane and how sharp the transition should be; this information predominantly comes from the training points in the transition region.

More formally, consider the minimization of the negative log likelihood loss function for this network, given by

$$f(x) = -\log \hat{p}(x) \qquad \text{with} \qquad \hat{p}(x) = \ell_5(\sigma(\alpha x - \beta)).$$

For the gradient, we need the derivative of the sigmoid function, $\sigma'_\gamma(x) = 2\ell'_\gamma(x)$ with

$$\ell'_\gamma(x) = \frac{\gamma e^{-\gamma x}}{(1 + e^{-\gamma x})^2} = \gamma \left(\frac{1 + e^{-\gamma x}}{(1 + e^{-\gamma x})^2} - \frac{1}{(1 + e^{-\gamma x})^2} \right) = \gamma[\ell_\gamma(x) - \ell^2_\gamma(x)],$$

and the derivative of the negative log of the logistic function:

$$\frac{d}{dx}\left[-\log(\ell_\gamma(x))\right] = -\frac{\ell'_\gamma(x)}{\ell_\gamma(f(x))} = -\gamma \frac{\ell_\gamma(x) - \ell^2_\gamma(x)}{\ell_\gamma(x)} = \gamma[\ell_\gamma(x) - 1].$$

Combining the above we have

$$\begin{aligned} \partial f/\partial\alpha &= \gamma\,[\ell_\gamma(\sigma(\alpha x-\beta))-1]\cdot\sigma'(\alpha x-\beta)\cdot x \\ \partial f/\partial\beta &= -\gamma\,[\ell_\gamma(\sigma(\alpha x-\beta))-1]\cdot\sigma'(\alpha x-\beta), \end{aligned}$$

with $\gamma = 5$. The loss function and partial derivatives with respect to α and β are plotted in Figure 9(b) and (c). The vertical lines in plot (c) indicate where the gradients fall below one percent of their asymptotic value. As expected, points beyond these lines do indeed contribute very little to the gradient, regardless of whether they are on the right or the wrong side of the hyperplane.

4.2 General mechanism

For the contribution of each sample to the gradient in general settings we need to take a detailed look at the backpropagation process. This is best illustrated using a concrete three-layer neural network:

$$\begin{aligned} A_1 &= \begin{bmatrix} 1.0 & 0.3 \\ 0.4 & -1.0 \end{bmatrix} \qquad b_1 = \begin{bmatrix} -1.0 \\ 0.5 \end{bmatrix} \qquad \nu_1 = \sigma_3, \\ A_2 &= \begin{bmatrix} -1 & -1 \\ 1 & -1 \\ 1 & 1 \\ -1 & 1 \end{bmatrix} \qquad b_2 = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \qquad \nu_2 = \sigma_3, \\ A_3 &= \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix} \qquad b_3 = \begin{bmatrix} -1.1 \\ -1.1 \end{bmatrix} \qquad \nu_3 = \mu. \end{aligned} \tag{8}$$

Denoting by $f(x)$ the negative log likelihood of $\mu(x)$, the forward and backward passes through the network can be written as

$$\begin{aligned} v_1 &= A_1 x_0 - b_1 \\ x_1 &= \sigma_3(v_1) & y_1 &= \frac{\partial x_1}{\partial v_1}\cdot z_2 = \sigma_3'(v_1)\cdot z_2 \\ v_2 &= A_2 x_1 - b_2 & z_2 &= \frac{\partial v_2}{\partial x_1}\cdot y_2 = A_2^T y_2 \\ x_2 &= \sigma_3(v_2) & y_2 &= \frac{\partial x_2}{\partial v_2}\cdot z_3 = \sigma_3'(v_2)\cdot z_3 \\ v_3 &= A_3 x_2 - b_3 & z_3 &= \frac{\partial v_3}{\partial x_2}\cdot y_3 = A_3^T y_3 \\ x_3 &= f(v_3) & y_3 &= \frac{\partial x_3}{\partial v_3} = \nabla f(v_3), \end{aligned} \tag{9}$$

where the left and right columns respectively denote the stages in the forward and backward pass. The regions formed during the forward pass are shown in Figure 10. With this, the partial differentials with respect to weight matrices and bias vectors are of the following form:

$$\frac{\partial f}{\partial [A_3]_{i,j}} = \frac{\partial v_3}{\partial [A_3]_{i,j}}\cdot y_3 = [x_2]_j\cdot[y_3]_i, \qquad \text{and} \qquad \frac{\partial f}{\partial b_3} = \frac{\partial v_3}{\partial b_3}\cdot y_3 = -y_3. \tag{10}$$

We now analyze each of the backpropagation steps to explain the relationship between the regions of high and low confidence at each of the neural network layers and the gradient values or importance of

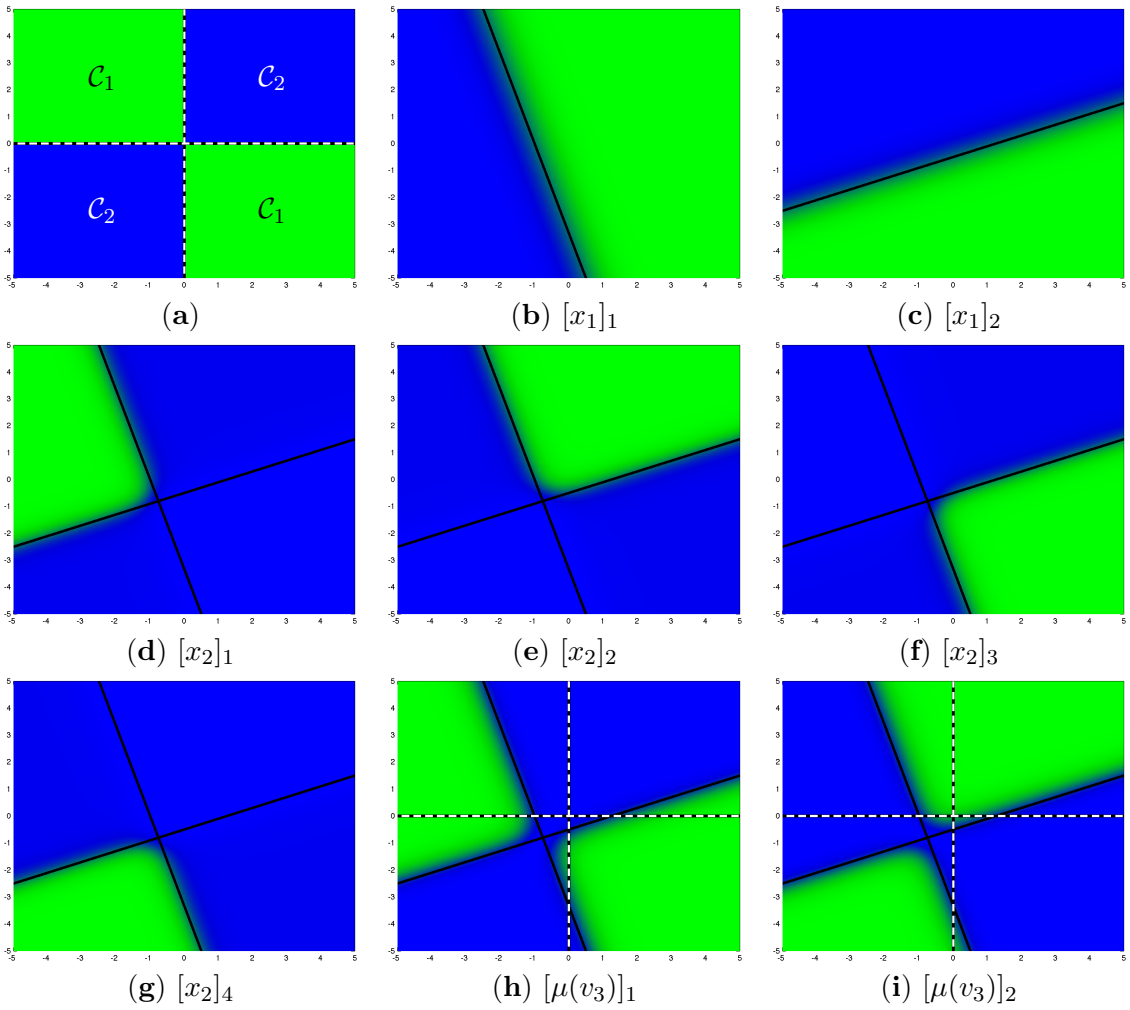
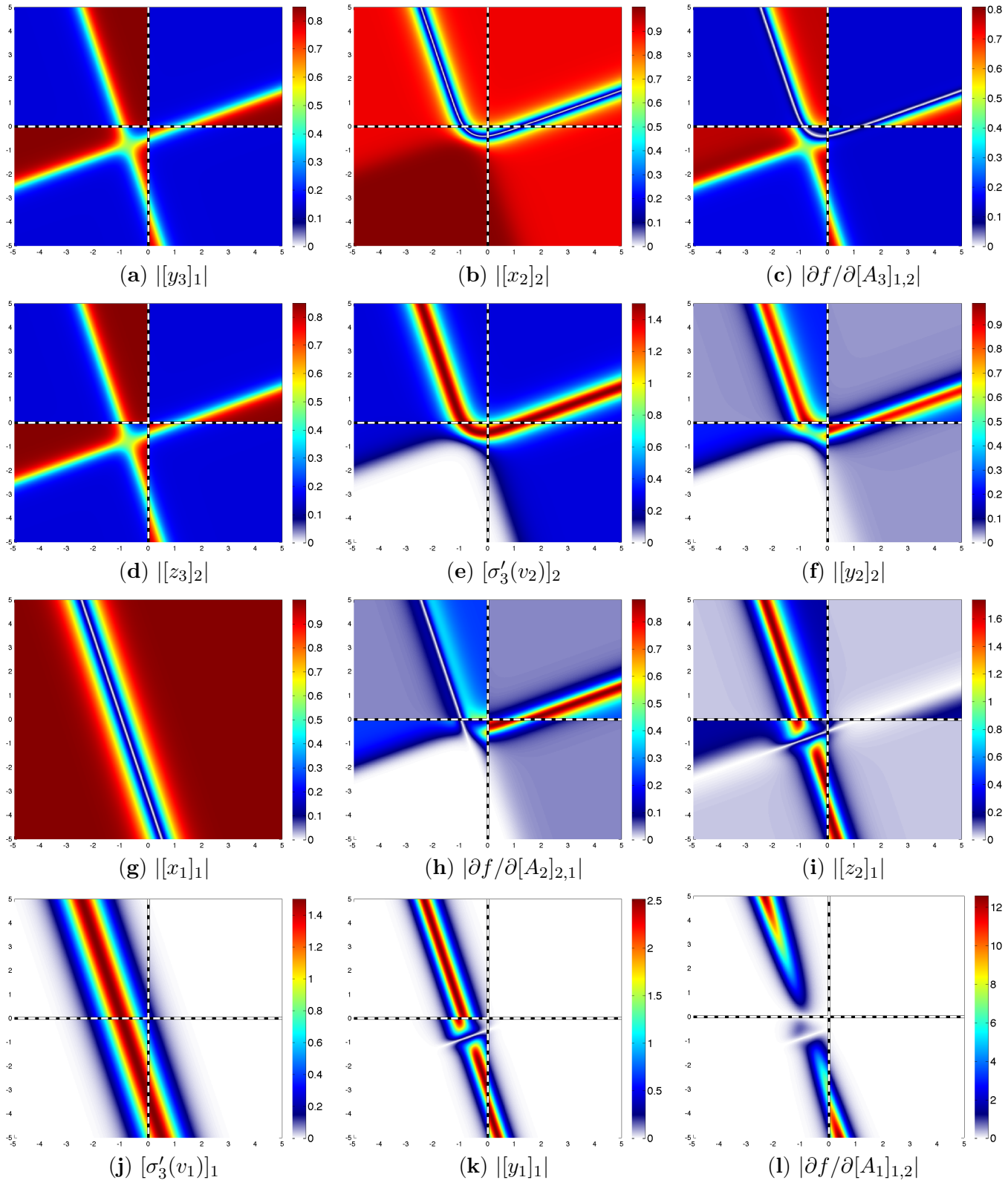


Figure 10: Regions corresponding to the neural network with weights defined by (8), with (a) the two ground-truth classes; (b,c) output regions of the first layer; (d–g) output regions of the second layer; and (h,i) output regions of the third layer (showing only the intermediate output $\mu(v_3)$ instead of the loss function output) with desired class boundaries superimposed.

different points in the feature space. In all plots we only show the absolute values of the quantities of interested because we are mostly interested in the relative magnitudes over the feature space rather than their signs. After a forward pass through the network we can evaluate the loss function and its gradients, shown in Figure 11(a). In this particular example we have $[y_3]_1 = -[y_3]_2$, so we only show the former. Given y_3 we can use (10) to compute the partial differentials of f with respect to the entries in A_3 and b_3. The partial differential with respect to b_3 simply coincides with $-y_3$, and is therefore not very interesting. On the other hand, we see that the partial differential with respect to $[A_3]_{i,j}$ is formed by multiplying $[y_3]_i$ with the output value $[x_2]_i$. When looking at the feature space representation for the specific case of $[A_3]_{1,2}$ and using absolute values, this corresponds to the pointwise multiplication of the values in Figure 11(a) with the mask shown Figure 11(b). This multiplication causes the partial differential to be reduced in areas of low confidence in $[x_2]_2$. In particular, it causes the partial differential to vanish at points at the zero crossing of the boundary regions, as illustrated by the white curve in the upper-right corner of Figure 11(c).



(**a**) $|[y_3]_1|$ (**b**) $|[x_2]_2|$ (**c**) $|\partial f/\partial[A_3]_{1,2}|$

(**d**) $|[z_3]_2|$ (**e**) $[\sigma_3'(v_2)]_2$ (**f**) $|[y_2]_2|$

(**g**) $|[x_1]_1|$ (**h**) $|\partial f/\partial[A_2]_{2,1}|$ (**i**) $|[z_2]_1|$

(**j**) $[\sigma_3'(v_1)]_1$ (**k**) $|[y_1]_1|$ (**l**) $|\partial f/\partial[A_1]_{1,2}|$

Figure 11: Illustration of the error backpropagation process.

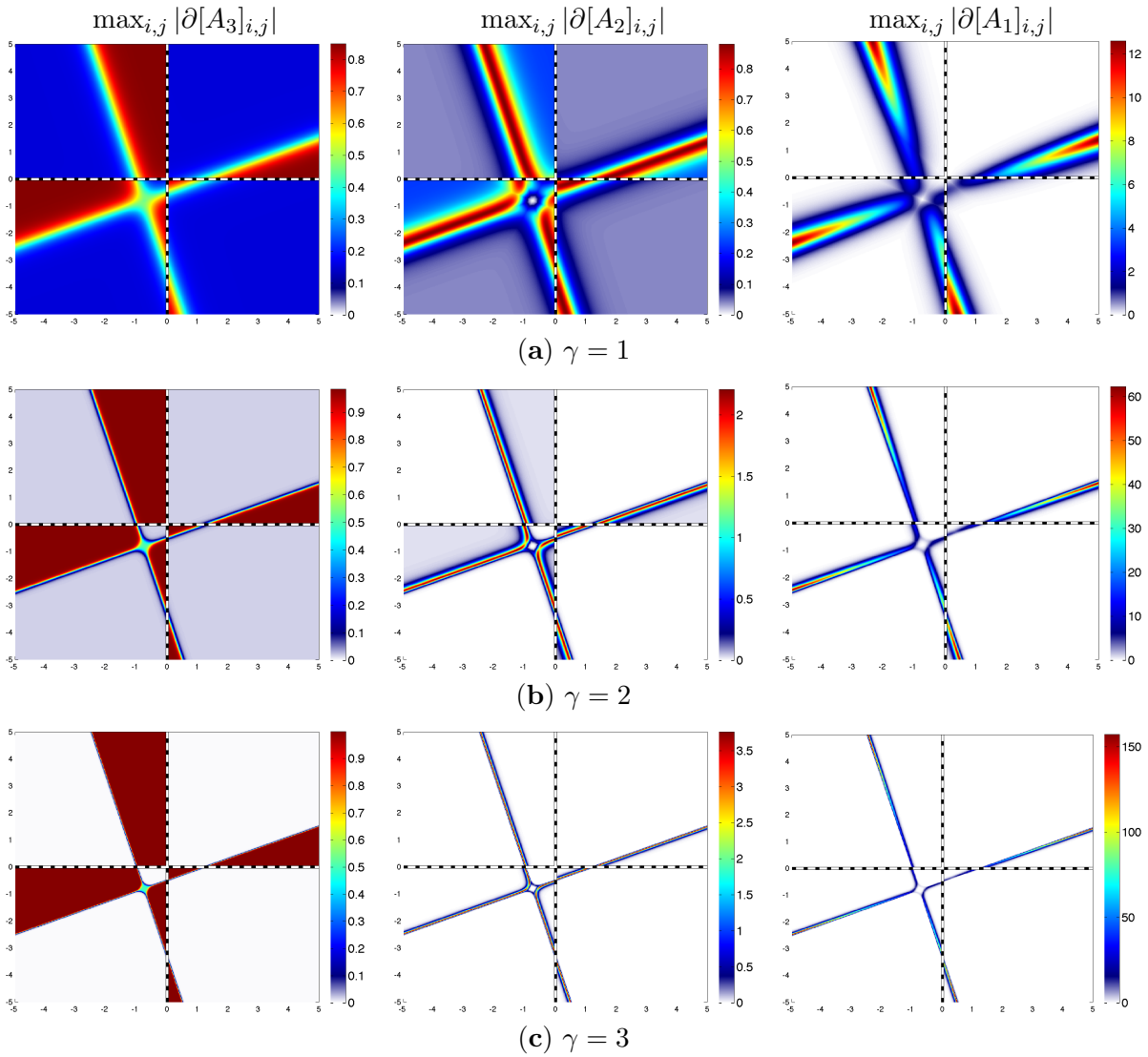
In the next stage of the backpropagation we multiply y_3 by the transpose of A_3 to obtain z_3, shown in Figure 11(d). As an intermediate value z_3 itself is not used, but it is further backpropagated though multiplication by $\sigma_3'(v_2)$ to obtain y_2. As illustrated in Figure 1(a), the gradient of the sigmoid is a kernel around the origin, and when applied to v_2, the preimage of x_2 under σ_3, it emphasizes the regions of low confidence and suppresses the regions of high confidence. This can be seen when comparing the mask for $[v_2]_2$, shown in Figure 11(e), with the corresponding region $[x_2]_2$ shown in Figure 10(d). The result obtained with multiplication by the mask is illustrated in Figure 11(f) and shows that backpropagation of the error is most predominant in the boundary region as well as in some regions where it was large to start with (most notably at the top of the bottom-left quadrant). From here we can compute the partial differential with respect to the entries of A_2 though multiplication by x_1, which again damps values around the transition region, and backpropagate further to get z_2, as shown in Figures 10(g)–(i). To obtain y_1, we need to multiply z_2 by the mask corresponding to the preimage of the regions in x_1. Unlike all other layers, these values are unbounded in the direction of the hyperplane normal and, as shown in Figure 10(j), result in masks that vanish away from the boundary region. Multiplication by the mask corresponding to $[v_1]_1$ gives $[y_1]_1$ shown in Figure 10(k). We finally obtain the partial differentials with respect to the entries in A_1 by multiplying by the corresponding entries in x_0. For the first layer this stage actually amplifies the gradient entries whenever the corresponding coordinate value exceeds one in absolute value. In subsequent layers the maximum values of x lie in the -1 to 1 output range of the sigmoid function and can therefore only reduce the resulting gradient components.

In Figure 12(a) we plot the maximum absolute gradient components for each of the three weight matrices. It is clear that the partial differentials with respect to A_3 are predominant in misclassified regions, but also exist outside of this region in areas where the objective function could be minimized further by increasing the confidence levels (scaling up the weight and bias terms). In the second layer, the backpropagated values are damped in the regions of high confidence and concentrate around the decision boundaries, which, in turn, are aligned with the underlying hyperplanes. Finally, in the first layer, we see that gradient values away from the hyperplanes have mostly vanished as a result of multiplication with the sigmoid gradient mask, despite the multiplication with potentially large coordinate values. Overall we therefore see the tendency of the gradients to become increasingly localized in feature space towards the first layer. The boundary shifts we discussed in Section 2.4 can lead to additional damping, as the sigmoid derivative masks no longer align with the peaks in the gradient field. Scaling of the weight and bias is detrimental to the backpropagation of the error (a phenomenon that is also known as saturation of the sigmoids [14]) and can lead to highly localized gradient values. This is illustrated in Figures 12(b) and (c) where we scale all weight and bias terms by a factor of 2 and 3, respectively. Especially in deep networks it can be seen that a single sharp mask in one of the layers can localize the backpropagating error and thereby affect all preceding layers. These figures also show that the increased scaling of the weights not only leads to localization, but also to attenuation of the gradients. In the first layer this is further aided by the multiplication with the coordinate values. Finally, we note that the above principles continue to hold for other loss functions as well.

5 Optimization

In the previous section we studied how individual training samples contribute to the overall gradient (7) of the loss function. In this section we take a closer look at the dynamic behavior and the changing relevance of training samples during optimization over the network parameter vector s. The parameter updates are gradient-descent steps of the form

$$s^{k+1} = s^k - \alpha \nabla \phi(s^k), \tag{11}$$



(**a**) $\gamma = 1$

(**b**) $\gamma = 2$

(**c**) $\gamma = 3$

Figure 12: Distribution over the feature space of the maximum gradient components for the weight matrices in each of the three layers. The weight and bias terms in all layers are scaled by a factor γ.

with learning rate α. The goal of this section is to clarify the relationships between the training set and the optimization process of the network parameters. To keep things simple we make no effort to improve the efficiency of the optimization process and, unless noted otherwise, we use a fixed learning rate with a moderate value of $\alpha = 0.01$. Likewise, we compute the exact gradient using the entire training set instead of using an approximation based on suitable chosen subsets, as is done in practically favored stochastic gradient descent (SGD) methods. Note, however, that the mechanisms exposed in this section are general enough to carry over to these and other methods without substantial changes. Similar findings may moreover apply to support vector machines and other models. Throughout this section we place a particular emphasis on the first layer of the neural network. To illustrate certain mechanisms it often helps to keep parameters of subsequent layers fixed. In this case it is implied that the corresponding entries in the gradient update in (11) are zeroed out.

5.1 Sampling density and transition width

To investigate the roles of sampling density and transition width we start with a very simple example with feature vectors $x \in [-6, 6] \times [-1, 1] \subset \mathbb{R}^2$, and two classes: one consisting of all points x in which the first entry is negative, and one consisting of all other points. Example training sets with samples in each of the two classes are plotted in Figure 13(a) and (b). Given such training sets we want to learn the classification using a neural network with a single hidden layer consisting of one node. To ensure that the classes are well defined we place four training samples—two for each class– near the interface of the two classes and sample the remaining points to the left and right of these points. Unless stated otherwise we keep all network parameters fixed except for the weights and bias terms in the first layer.

5.1.1 Sampling density

In the first experiment we study how the number or density of training points affects the optimization. We initialize the network with parameters

$$A_1 = \tfrac{25}{\sqrt{1.09}}[1, 0.3],\ b_1 = A_1 \cdot [2, 0]^T, \quad A_2 = [3; -3],\ b_2 = [0; 0],$$

and keep the parameters in the second layer fixed. Parameter b_1 is chosen such that the initial hyperplane goes through the point $(2, 0)$. Training sets consist of n samples, including the four at the interface, and are chosen such that the number of points in each class differs by at most one. Figures 13(a) and (b) illustrate such sets for $n = 50$, and $n = 800$, respectively. The hyperplane is indicated by a thick black line, bordered with two dashes lines which indicate the location where the output of the first layer is equal to ± 0.95. Figure 13(c) shows the magnitude of the partial differential with respect to $[A_1]_1$ at the initial parameter setting over the entire domain. The gradient $\nabla\phi(s)$ is then computed as the average of the gradient values evaluated at the individual training samples. As a measure of progress we can look at the area of the misclassified region, i.e., the region between the y-axis and the hyperplane. Figure 13(e) shows this area as a function of iteration for different sampling densities. The shape of the loss function curves are very similar to these and we therefore omit them here. For $n = 800$ and $n = 3200$ the area of the misclassified region steadily goes down to zero, although the rate at which it does so gradually diminishes. Although not apparent from the curves, this phenomenon happens for all the training sets used here and we will explain exactly why this happens in Section 5.2. Progress for $n = 50$ and $n = 200$ appears much less uniform and exhibits pronounced stages of fast and slow progress. The reason for this is a combination of the sampling density and the localized gradient. From Figure 13(c) we can see that the gradient field is concentrated around the hyperplane, with peak values slightly to the left of the hyperplane. When the sampling density is low it may happen that none of the training samples is close to the hyperplane. When this happens, the gradient will be small, and consequently progress will be slow. When one or more points are close to the hyperplane, the gradient will be larger and progress is faster. Figure 13(f) shows the rate of change in the area of the misclassified region along with the distance between the hyperplane and its nearest training sample for $n = 50$. It can be seen that the rate increases as the hyperplane moves towards the training sample, with the peak rate happening just before the hyperplane reaches the point. After that the rate gradually drops again as the hyperplane slowly moves further away from the sample. This is precisely the state at 300,000 iterations, which is illustrated in Figure 13(d). For $n = 25$, we find ourselves in the same situation right at the start. Initially we move away from a single training point, but as a consequence of the low sampling density, no other sampling points are nearby, causing a prolonged period of very slow progress. The discrete nature of training samples is less pronounced when the overall sampling density is high, or when the transition widths are large.

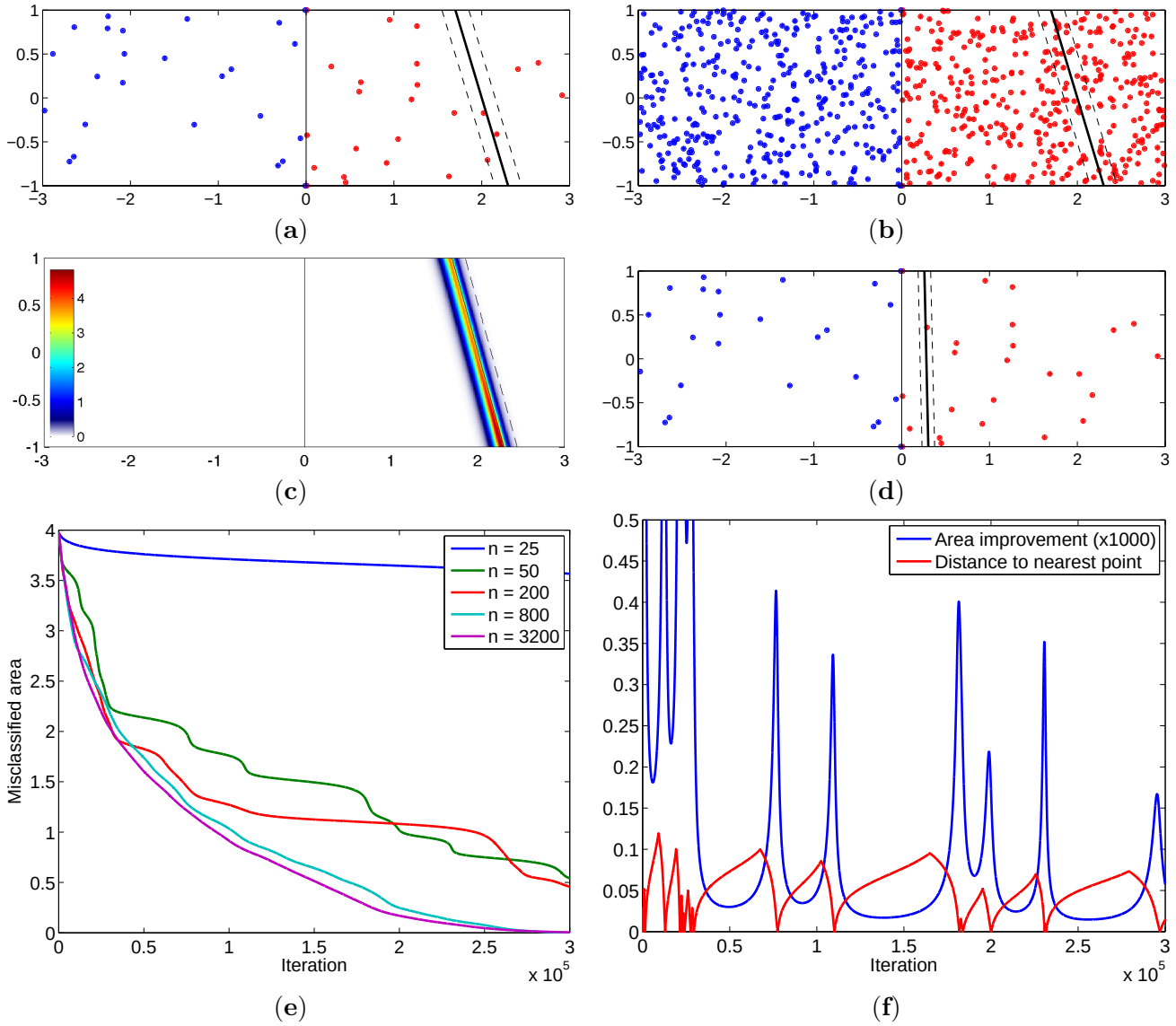


Figure 13: Simple domain and initial hyperplane location with (a) 50, and (b) 800 training samples equally divided over two classes, along with (c) the initial gradient field. Plot (d) shows the location of the hyperplane after 300,000 iterations and (e) shows the area of the misclassified region as a function of iteration for different numbers of training samples. Plot (f) shows the reduction in misclassified area per iteration and the distance between the hyperplane and the nearest sample when 50 training samples are used.

5.1.2 Transition width

To illustrate the effect of transition widths, we used the setting with 3,200 samples as described above, but scaled the row vector of the initial A_1 to have Euclidean norm ranging from 1 to 100. In each case we adjust b_1 such that the initial hyperplane goes through the point $(2, 0)$. As shown in Figure 14(a), the misclassified area reaches zero almost immediately when A_1 is scaled to have unit norm. In other words, the hyperplane is placed correctly in this case after only 3,260 iterations. As the norm of the initial A_1 increases, it takes longer to reach this point: for an initial norm of 10 it takes some 72,580 iterations, whereas for an initial norm of 25 it takes over 300,000. Accordingly, we see from Figure 14(b) that the loss also drops much faster for small weights than it does for large weights. However, once the hyperplane is in place, the only way to decrease the loss is by scaling the weights to improve the confidence. This process can be somewhat slow when the weights are small and the hyperplane placement is finalized (as is the case when we start with small initial weights). As a result, the setup with initial weight of 25 eventually catches up with the earlier two, simply because it has a much sharper transition at the boundary as the hyperplane finally closes in to the right location.

The reason why the hyperplane moves faster for small initial weights is twofold. First, the transition width and support of the gradient field are larger. As a result, more sample points contribute to the gradient, leading to a larger overall gradient value. This is shown in Figures 14(c) and (d) in which we plot the norm of the gradients with respect to A_1 and b_1 when choosing $A_1 = [a, 0]$, and b such that the hyperplane goes through the given location on the x-axis. The gradients with respect to either parameters are larger for smaller a. (Unlike in Figure 12, the localization of the gradient here is due only to scaling of the weights in the first layer; the intensity of the gradient field therefore remains unaffected.) As the value of a increases, the curves in Figures 14(c) before more linear. For those values the gradient is highly localized and, aside from the scaling by the training point coordinates, largely independent of the hyperplane location. The gradient with respect to b_1 does not include this scaling and therefore remains nearly constant as long as the overlap between the transition width and the class boundary is negligible. As the hyperplane moves into the right place, the gradient vanishes due to the cancellation of the contributions from the training points from the classes on either side of it. The curves for $a = 16$ and, to a lesser extent, for $a = 8$ show minor aberrations due to a relatively low sampling density compared to the transition width. Second, having larger gradient values for smaller weights means that the relative changes in weights are amplified, allowing the hyperplane to move faster.

5.2 Controlling the parameter scale

In this section we work with a modified version of the domain shown in Figure 13(a). In particular, we change the horizontal extent from $[-3, 3]$ to $[-30, 30]$, and randomly select 500 training samples uniformly at random and equally divided over the two classes (thus leaving the sampling density unaffected). As a first experiment we optimize a three-layer network with initial parameters:

$$A_1 = [1, 0.3]/\sqrt{1.09}, \quad b_1 = A_1 \cdot [25; 0], \qquad A_2 = 3, \quad b_2 = 0, \qquad \text{and} \qquad A_3 = [3; -3], \quad b_3 = 0. \tag{12}$$

When we look at the row-norms of the weight matrices, plotted in Figure 15(a), we can see that all of them are growing. Such growth is perfectly fine when improving the final confidence levels, but can be detrimental during the optimization process. Indeed, we can see from Figure 15(b) that the hyperplane never quite reaches the origin, despite the large number of iterations. As illustrated in Figure 12, scaling of the weight and bias terms leads to increasingly localized gradients. When the training sample density is low compared to the size of the regions where the gradient values are significant, it can easily happen that no significant values from the gradient field are sampled into the gradient. This applies in particular to the first several layers (depending on the network depth) where the gradient fields become increasingly

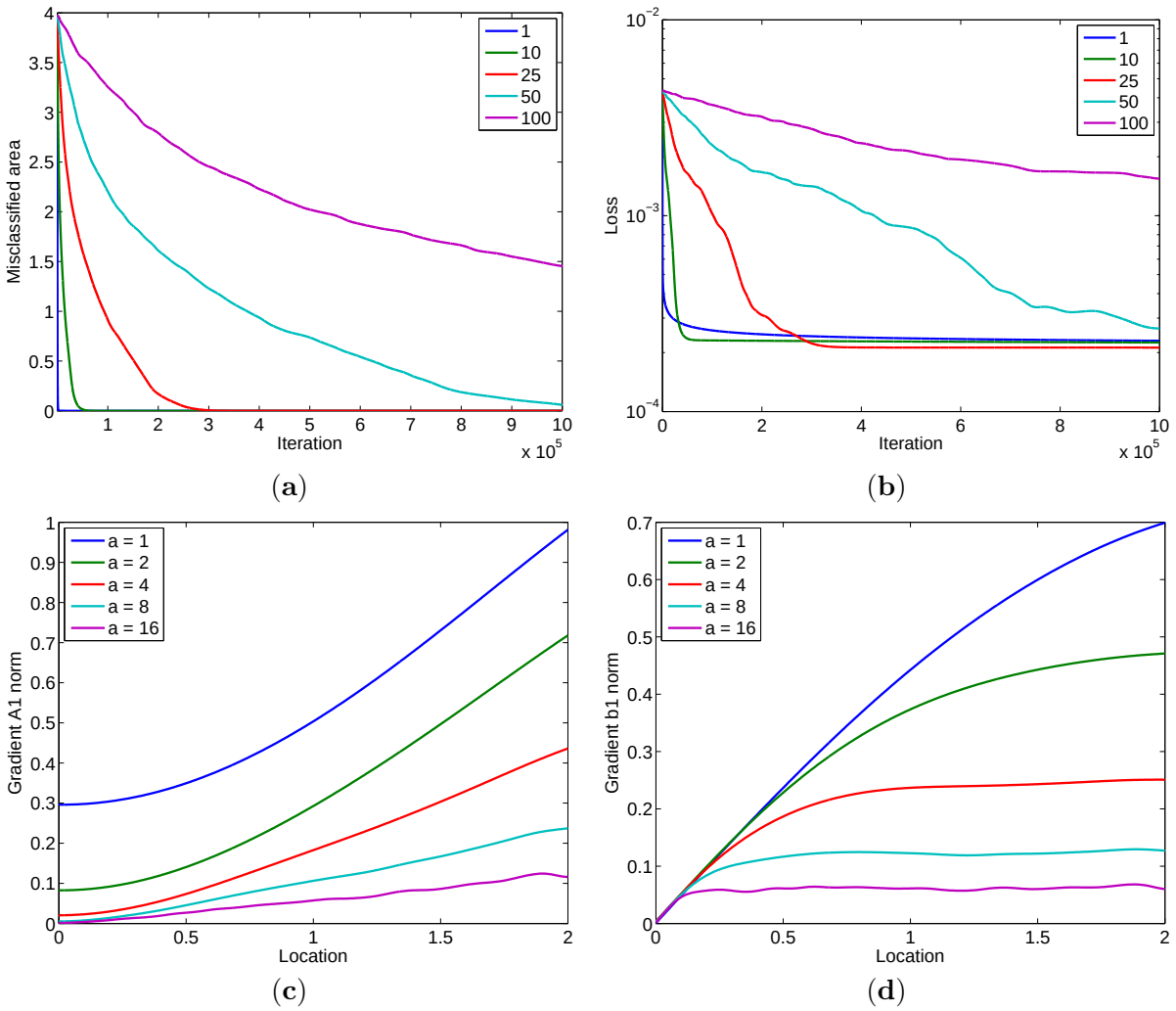


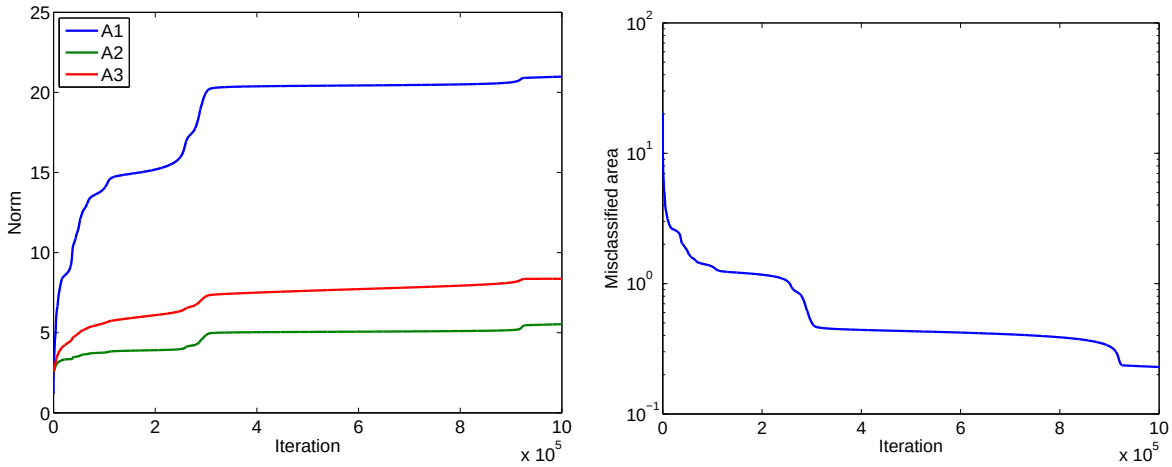
Figure 14: Plots of (a) misclassified area and (b) the value of the loss function as a function of iterations for different initial weights. Norms of the gradients with respect to (c) A_1 and (d) b_1 as a function of hyperplane location with $A_1 = [a, 0]$ for different values of a.

localized (though not necessarily small) because of the sigmoidal gradient masks that are applied during back propagation, along with shifts in the boundary regions. This 'vanishing gradient' phenomenon can prematurely bring the training process to a halt; not because a local minimum is reached, but simply because the sampled gradient values are excessively small. Scaling of the parameters in any layers except the last can cause the gradient field to become highly localized for the current and all preceding layers. This can cause a cascading effect in which suboptimal parameters in a stalled first layer lead to further parameter scaling in later layers, eventually causing the second layer to stall, and so on. To avoid this, we need to control the parameter scale during optimization.

Parameter growth can be controlled by adding a regularization or penalty term to the loss function, or by imposing explicit constraints. Extending (4) we could use

$$\underset{s}{\text{minimize}} \quad \phi(s) + r(s), \qquad \text{or} \qquad \begin{aligned} \underset{s}{\text{minimize}} \quad & \phi(s) \\ \text{subject to} \quad & c_i(s) \leq 0, \end{aligned} \tag{13}$$

where $r(s)$ is a regularization function, and $c_i(s)$ are constraint functions. The discussions so far suggest



(**a**) (**b**)

Figure 15: Plots of (a) growth in the norms of the weight matrices and (b) reduction of the misclassified area as a function of iteration.

some natural choices of functions for different layers. The function in the first layer should generally be based on the (Euclidean) ℓ_2 norm of each of the rows in A_1, such as their sum, maximum, or ℓ_2 norm. The reason for this is that each row in A_1 defines the normal of a hyperplane, and using any function other than an ℓ_2 norm may introduce a bias in the hyperplane directions. For subsequent layers k (except possibly the last layer) we may want to ensure that the output cannot be too large. In the worst case, each input from the previous layer is close to $+1$ or -1, and we can limit the output value by ensuring that the sum of absolute values, i.e., the ℓ_1 norm, of each row in A_k is sufficiently small. Of course, the corresponding value in b_k could still be large, which may suggest adding a constraint that $\|[A_k]_j\|_1 \leq |[b_k]_j|$ for each row j. However, this constraint is non-convex and may impede sign changes in b. The use of an ℓ_1 norm-based penalty or constraint on intermediate layers has the additional benefit that it leads to sparse weight matrices, which can help reduce model complexity as well as evaluation cost.

As an illustration of the effect of ℓ_2 regularization on the first layer we consider the setting as given in (12), but with the second layer removed. We optimize the weight and bias terms in the first layer using the standard formulation (4), as well as those in (13) with $r(s) = \lambda/2\|A_1^T\|_2^2$ or $c(s) = \|A_1^T\|_2 \leq \kappa$. For simplicity we keep all other network parameters fixed. Optimization in the constrained setting is done using a basic gradient projection method with step size fixed to 0.01, as before. The results are show in Figure 16. When using the standard formulation we see from Figure 16(a) that, like above and in Figures 13(a,d), the ℓ_2 norm of the row in A_1 keeps growing. This is explained as follows: suppose the hyperplane is vertical with A_1 of the form $[a, 0]$, and $b_1 = b$. Then the area of the misclassified region is $2|b|/|a|$. We can therefore reduce the misclassified area (and in this case the loss function) by increasing a and decreasing b, which is exactly what happens. However, from Figure 16(b) we can see that the rate at which the misclassified area is reduced decreases. The reason for this is a combination of three factors. First, the speed at which $|b|/|a|$ goes towards zero slows down as a gets larger. Second, the peak of the gradient field lies along the hyperplane and shifts towards the origin with it. Because the gradient in the first layer is formed by a multiplication of the backpropagated error with the feature vectors (coordinates), the gradient gets smaller too. Third, because of the growing norm of A_1, the transition width shrinks and causes the gradient to become more localized. As a result, fewer training points sample the gradient field at significant values, leading to smaller overall gradients with respect to both A_1 and b_1. As an aside, note that if the class boundary were to the left of the current starting point, all the above three effects

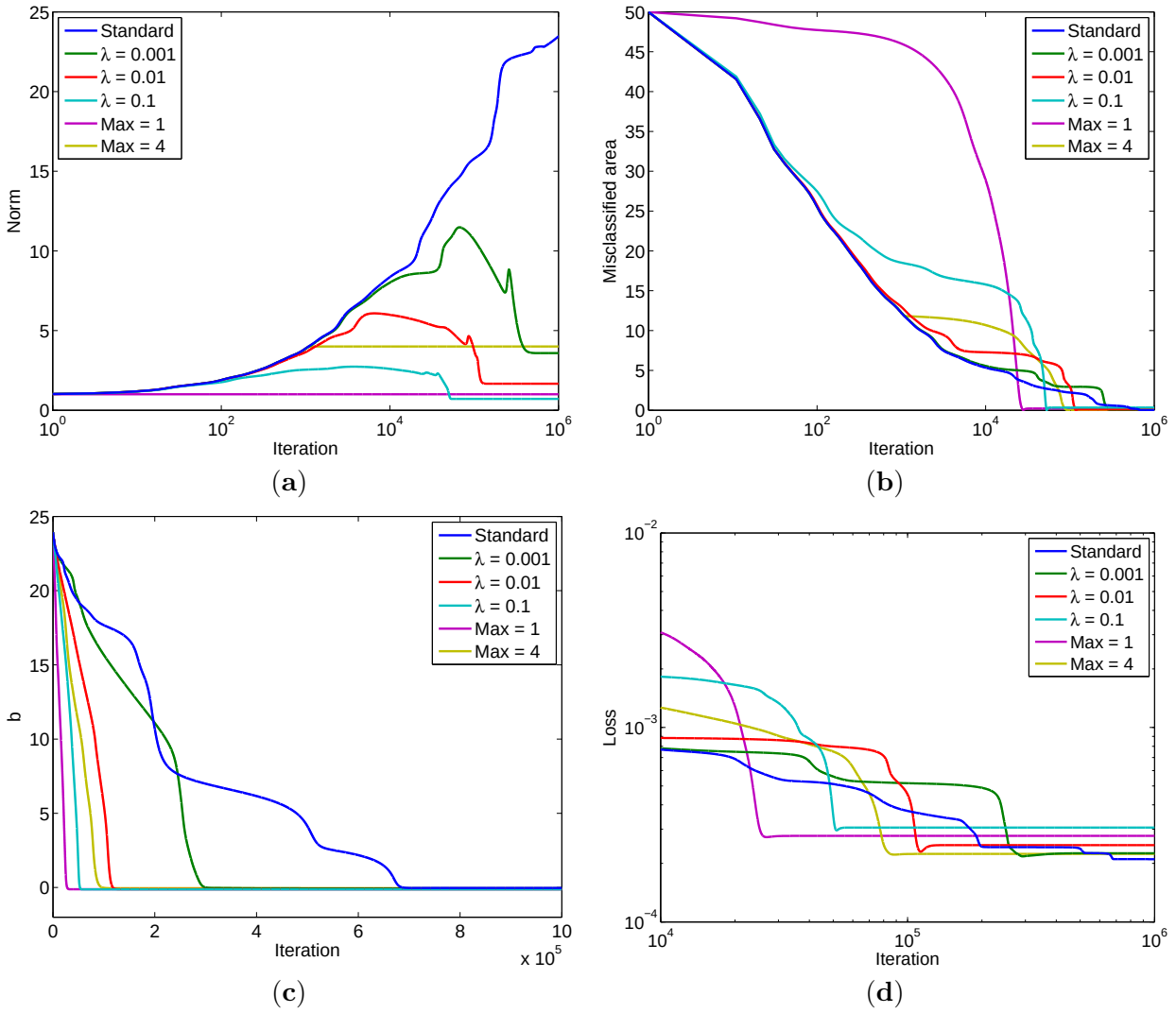


Figure 16: Differences between (a) the norm of A_1; (b) the area of the misclassified region; (c) the magnitude of b; and (d) the loss function, as a function of iteration for standard gradient descent and variations of regularized and constrained optimization.

would be beneficial and help speed up convergence to the right location.

There is not much we can be do about the first two causes, but adding a regularization term or imposing constraints, certainly does help with the third, and we can see from Figure 16(a) that the norm of A_1 indeed does not grow as much as in the standard approach. At first glance, this seems to hamper the reduction of the misclassified area, shown in Figure 16(b). This is true initially when most of the progress is due to the scaling of A_1, however, the moderate growth in A_1 also prevents strong localization of the gradient and therefore results in much steadier reduction of b, as shown in Figure 16(c). The overall effect is that the constrained and regularized methods catch up with the standard method and reduce the misclassified area to zero first. Even so, when looking at the values of the loss function without the penalty term, as plotted in Figure 16(d), we see that the standard method still reaches the lowest value, even though all methods have zero misclassification. As before, this is because the two classes are disjoint and are best separated with a very sharp transition. The order in which the lines in Figure 16(d)

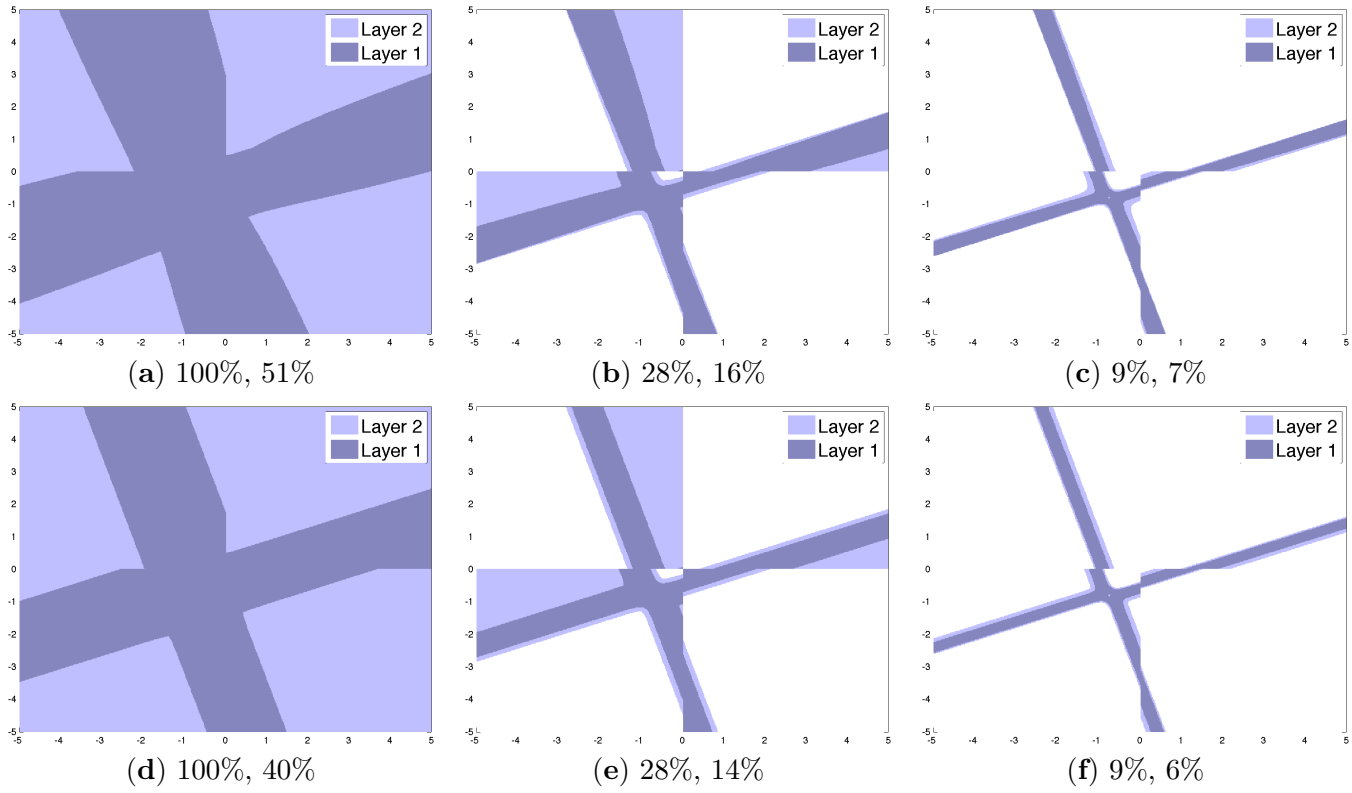
appear at the end, is therefore related to the norms in Figure 16(a). This suggests the use of cooling or continuation strategies in which norms are gradually allowed to increase. The initial small weights ensure that many of the training samples are informative and contribute to the gradients of all layers, thereby allowing the network to find a coarse class alignment. From there the weights can be allowed to increase slowly to fine tune the classification and increase confidence levels. Of course, while doing so, care needs to be taken not to excessively scale the weights as this can lead to overfitting.

Instead of scaling weight and bias terms we could also consider scaling sigmoid parameters γ, or learn them [21]. One interesting observation here is that even though all networks with parameters αA_1, αb_1, and γ/α are equivalent for $\alpha > 0$, their training certainly is not. The reason is the $1/\alpha$ term that applies to the gradients with respect to A_1 and b_1. Choosing $\alpha > 1$ means larger parameter values and smaller gradients. This reduces both the absolute and relative change in parameter values and is equivalent to having a stepsize that is α^2 smaller. Instead of doing joint optimization over both the layer and nonlinearity parameters, it is also possible to learn the nonlinearity parameters as a separate stage after optimization of the weight and bias terms.

5.3 Subsampling and partial backpropagation

Consider the scenario shown in Figure 13(a) and suppose we double the number of training samples by adding additional points to the left and right of the current domain. In the original setting, the gradient with respect to the weights in the first layer is obtained by sampling the gradient field shown in Figure 13(c). In the updated setting, all newly added points are located away from the decision boundary. As a result, their contribution to the gradient is relatively small and the overall gradient may be very similar to the original setting. However, because the loss function $\phi(s)$ in (4) is defined as the average of the individual loss-function components, we now need to divide by $2N$ rather than N, thereby effectively scaling down the gradient by a factor of approximately two. Another way to say this is that the stepsize is almost halved by adding the new points. This example is of course somewhat contrived, since additional training samples can typically be expected to follow the same distribution as existing points and therefore increase sampling density. Nevertheless, this example may make us wonder whether the training samples on the left and right-most side of the original domain are really needed; after all, using only the most informative samples in the gradient essentially amounts to larger stepsize and possibly a reduction in computation.

For sufficiently deep networks with even moderate weights, the hyperplane learning is already rather myopic in the sense that only the training points close enough to the hyperplane provide information on where to move it. This suggests a scheme in which we subsample the training set and for one or more iterations work with only those points that are relevant. We could for example evaluate $v_1 = A_1 x_0 - b_1$ for each input sample x_0, and proceed with the forward and backward pass only if the minimum absolute entry in v_1 is sufficiently small (i.e., the point lies close enough to at least one of the hyperplanes). This approach works to some extend for the first layer when the remaining layers are kept fixed, however, it does not generalize because the informative gradient regions can differ substantially between layers (see e.g., Figure 12). Instead of forming a single subsampled set of training points for all layers we can also form a series of sets—one for each layer—such that all points in a set contribute significantly to the gradient for the corresponding and subsequent layers. This allows us to appropriately scale the gradients for each layer. It also facilitates partial backpropagation in which the error is backpropagated only up to the relevant layer, thereby reducing the number of matrix-vector products. Given a batch of points, we could determine the appropriate set by evaluating the gradient contribution to each layer and finding the lowest layer for which the contribution is above some threshold. Alternatively, we could use the following partial backpropagation approach, which may be beneficial in its own right, especially for deep networks.



(**a**) 100%, 51% (**b**) 28%, 16% (**c**) 9%, 7%

(**d**) 100%, 40% (**e**) 28%, 14% (**f**) 9%, 6%

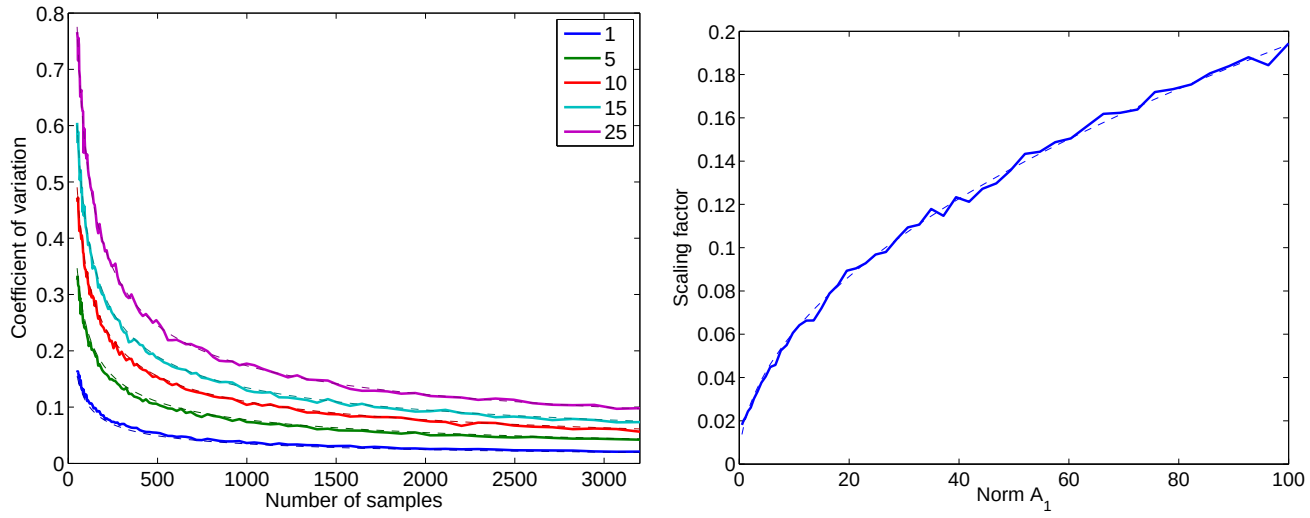
Figure 17: Regions of the feature space that are backpropagated to layers 2 and 1. From left to right we have the settings $\gamma = 1$, $\gamma = 2$, and $\gamma = 3$ from Figure 12, respectively. The top row shows the results obtained with the Frobenius norm of the gradients with respect to the weight matrices in each layer. The bottom row shows the results obtained by bounding the gradients elementwise. The percentages indicate the fraction of the feature space that was backpropagated to the second, and first layer.

In order to do partial backpropagation, we need to determine at which layer to stop. If this information is not given a priori, we need a conservative and efficient mechanism that determines if further backpropagation is warranted. One such method is to determine an upper bound on the gradient components of all layers up to the current layer and decide if this is sufficiently small. We now derive bounds on $\|\partial f/\partial A_k\|_F$ and $\|\partial f/\partial b_k\|_F$ as well as on $\max_{i,j} |[\partial f/\partial A_k]_{i,j}|$ and $\|\partial f/\partial b_k\|_\infty$. It easily follows from (10) that these quantities are equal to $\|x_{k-1}\|_2\|y_k\|_2$ and $\|y_k\|_2$, respectively $\|x_{k-1}\|_\infty\|y_k\|_\infty$ and $\|y_k\|_\infty$. Since x_{k-1} is known explicitly from the forward pass, it suffices to bound the norms of y_k. In fact, what we are really after is to bound the norms of y_k for all $1 \le k < j$ given y_j, since we can stop backpropagation only if all of them are sufficiently small. For the ℓ_2 norm we have

$$\|y_{k-1}\|_2 \;\le\; \|\sigma'_{\gamma_{k-1}}(v_{k-1})\|_\infty \|z_k\|_2 \;\le\; \sigma'_{\gamma_{k-1}}([v_{k-1}]_i) \cdot \sigma_{\max}(A_k)\|y_k\|_2, \tag{14}$$

where $i := \arg\min_j |[v_{k-1}]_j|$, and $\sigma_{\max}(A_k)$ is the largest singular values of A_k. Once we have a bound on $\|y_j\|_2$ we can apply (14) with $k = j$ to bound $\|y_{j-1}\|_2$. Although computation of $\sigma_{\max}(A_k)$ needs to be done only once per batch but may still be prohibitively expensive. In practice, however, it may suffice to work with an approximate value, or use an alternative bound instead. For ℓ_∞ we find

$$\|y_{k-1}\|_\infty \;\le\; \max_i \{\sigma'_{\gamma_{k-1}}([v_{k-1}]_i) \cdot \|[A_k]_i\|_2 \|y_k\|_2\} \;\le\; \|\sigma'_{\gamma_{k-1}}(v_k)\|_\infty \|y_k\|_2 \max_i \{\|[A_k]_i\|_2\}, \tag{15}$$



(**a**) (**b**)

Figure 18: Plots of (a) the coefficient of variation as a function of the number of training samples N with curves of the form $c(w)/\sqrt{N}$; and (b) scaling factor $c(w) \approx 1.1\sqrt{w}$ as a function of w, the norm of A_1. Dashed lines show the least-square fits of the above forms.

where the second, looser bound can be used if we want to avoid evaluating $\sigma'_{\gamma_{k-1}}$ for all entries in v_k; the infinity norm of this vector can be evaluated as above.

We applied the second bound in (14) and the first bound in (15) to the setting for Figure 12 as follows. We first compute y_3 and evaluate the bound the gradients with respect to the weight and bias terms in the first and second layer. If these bounds are smaller than 0.05 and 0.01, respectively, we stop backpropagation. Otherwise, we evaluate y_2 and update the bound on the gradient with respect to the parameters of the first layer. If this is less than 0.05 we stop backpropagation, otherwise we evaluate y_1 and complete the backpropagation process. In Figure 17 we show the regions of the feature space where backpropagation reaches the second, respectively first layer. These regions closely match the predominant regions of the gradient fields shown in Figure 12. In practical applications the threshold values could be based on previously computed (partial) gradient values, and may be adjusted when the number of training samples that backpropagate to a given layer falls below some threshold.

5.4 Sampled gradient variance

One of the most successful techniques in neural network optimization is mini-batch stochastic gradient descent. In this method the gradient is approximated by averaging the gradient contribution from randomly selected subsets of the training data. The advantage of this approach is that one pass over the data leads to numerous descent iterations rather than just one, thereby enabling much faster progress. The approximate gradients used in stochastic gradient descent match the exact gradient only in expectation, and there is an inherent trade-off between the computational complexity and the variance in the sampled gradients. Increasing the number of samples per batch lowers the variance, but increases the computational complexity and additionally reduces the number of parameter updates per epoch.

The variance does not only depend on the batch size, but also depends on the weight: when the gradient becomes very localized, we can expect the variance of a sampled gradient to increase. To verify this we again used the simple domain as in Figure 13(a) where A_1 was scaled to have different norms. For each scaling factor we repeatedly sampled the gradient with respect to A_1 at batches of various sizes

with training points selected uniformly at random from the domain. The resulting coefficients-of-variation curves (standard deviation over the mean) are plotted in Figure 18(a). As expected, the curves are of the form $c(w)/\sqrt{N}$, where N is the number of training samples and $c(w)$ is a scaling factor that depends on w, the norm of A_1. From this it is clear that the larger the norm of A_1, the larger the coefficient of variation. To get an idea about the scaling factors for this problem we determined the standard deviation in the gradient for a range of w values, using $3,200$ samples. The dashed line in Figure 18(b) shows that the scaling coefficients $c(w)$ for the current setting are well approximated by $1.1\sqrt{w}$. Although the exact nature of the scaling coefficients may differ in other setting, this example does clearly illustrate that the batch size required to maintain a certain variance level depends on the scaling of the weights. As the weights increase (for example towards the end of the optimization process) it is therefore necessary to increase the batch size to achieve an appropriate level of variance.

6 Conclusions

We reviewed and studied the decision region formation in feedforward neural networks with sigmoidal nonlinearities. Although the definition of hyperplanes and their subsequent combination is well known, very little attention has so far been given to transitions regions at the boundaries of classes and other regions with varying levels of classification confidence. We clarified the relation between the scaling of the weight matrices, the increase in confidence and sharpening of the transition regions, and the corresponding localization of the gradient field. The degree of localization differs per layer and is one of the main factors that determine how much progress can be made at each step of the training process: a high level of localization combined with a relatively coarse sampling density or small batch size can lead both to large variations in the gradient and to the vanishing gradient problem where updates to one or more layers become excessively small. The gradient field tends to become increasingly localized towards the first layer, and the parameters in this layer are therefore most likely to get stuck prematurely. When this happens, subsequent layers must form classifications regions based on suboptimal hyperplane locations. It is often possible to slightly decrease the loss function by increasing confidence levels by scaling parameters in later layers. This can lead to a cascading effect in which layers successively get stuck. The use of regularized or constrained optimization can help control the scaling of the weights, thereby limiting the amount of gradient localization and thus avoiding or reducing these problems. By gradually allowing the weights to increase it is possible to balance progress in the learning process, and attaining decision regions with sufficiently high confidence levels. In addition, regularized and constrained optimization can help prevent overfitting. Analysis of the gradient field also shows that at any given iteration, the contributions of different training points to the gradient can vary substantially. Localization of the gradient towards the first layer also means that some points are informative only from one layer onwards. Together this suggests dynamic subset selection and partial backpropagation, or adaptive selection of the step size for each layer depending on the number of relevant points.

We hope that some of the results presented in this paper will contribute to a better understanding of neural networks and eventually lead to new or improved algorithms. There remain several topics that are interesting but beyond the scope of the present paper. For example, it would be interesting to see what the hyperplanes generated during pre-training using restricted Boltzmann machines [10] look like, and if there are better choices. One possible option is to select random training samples from each class and generate randomly oriented hyperplanes through these points by appropriate choice of b. Likewise, given a hyperplane orientation and a desired class, it is also possible to place the hyperplane at the class boundary by choosing b to coincide with the largest or smallest inner product of the normal with points from that class. Another interesting topic is an extension of this work to other nonlinearities such as the currently popular rectified linear unit given by $\nu(x) = \max(0, x)$. The advantage of these units is that

gradient masks are far less localized, which allows the error to backpropagate more easily and therefore reduces gradient localization. It would be interesting to look at the mechanisms involved in the formation of decision regions, which differ from those of sigmoidal units. For example, it is not entirely clear how the logical AND should be implemented: summing inverted regions and thresholding may work in some cases, but more generally it should consist of the minimum of all input regions. In terms of combinatorial properties, bounds on the number of regions generated using neural networks with rectified and piecewise linear functions were recently obtained in [16, 18]. The main problem with rectified linear units is that it maps all negative inputs to zero, thereby creating a zero gradient mask at those locations. The softplus nonlinearity [8], which is a smooth alternative in which the gradient mask never vanishes, would also be of interest. Finally it would be good to get a better understanding of dropout [11] and second-order methods from a feature-space perspective.

References

[1] Martin Anthony. Boolean functions and artificial neural networks. Technical Report CDAM research report series, LSE-CDAM-2003-01, Centre for Discrete and Applicable Mathematics, London School of Economics and Political Science, London, UK, 2003.

[2] Christopher M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, Inc., New York, NY, USA, 1995.

[3] Ralph P. Boland and Jorge Urrutia. Separating collections of points in Euclidean spaces. *Information Processing Letters*, 53(4):177–183, February 1995.

[4] Efim M. Bronshteyn and L. D. Ivanov. The approximation of convex sets by polyhedra. *Sibirian Mathematical Journal*, 16(5):852–853, 1975.

[5] Efim M. Bronstein. Approximation of convex sets by polytopes. *Journal of Mathematical Sciences*, 153(6):727–762, 2008.

[6] Gerald H. L. Cheang and Andrew R. Barron. A better approximation for balls. *Journal of Approximation Theory*, 104(2):183–203, 2000.

[7] Richard M. Dudley. Metric entropy of some classes of sets with differentiable boundaries. *Journal of Approximation Theory*, 10(3):227–236, 1974.

[8] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier networks. In *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics*, volume 15, pages 315–323. JMLR W&CP, 2011.

[9] Geoffrey Hinton, Li Deng, Dong Yu, George E. Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N. Sainath, and Brian Kingsbury. Deep neural networks for acoustic modeling in speech recognition. *IEEE Signal Processing Magazine*, 29(6):82–97, November 2012.

[10] Geoffrey E. Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18:1527–1554, 2006.

[11] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *The Computing Resarch Repository (CoRR)*, abs/1207.0580, 2012.

[12] William Y. Huang and Richard P. Lippmann. Neural net and traditional classifiers. In D. Z. Anderson, editor, *Neural Information Processing Systems*, pages 387–396, 1988.

[13] Károly Böröczky Jr. and Gergely Wintsche. Covering the sphere by equal spherical balls. In B. Aronov, S. Bazú, M. Sharir, and J. Pach, editors, *Discrete and Computational Geometry – The Goldman-Pollak Festschrift*, pages 237–253. Springer, 2003.

[14] Yann LeCun, Lean Bottou, Genevieve B. Orr, and Klaus-Robert Müller. Efficient backprop. In Genevieve B. Orr and Klaus-Robert Müller, editors, *Neural Networks: Tricks of the Trade*, volume 1524 of *Lecture notes in computer science*, pages 9–50. Springer, 1998.

[15] John Makhoul, Richard Schwartz, and Amro El-Jaroudi. Classificadtion capabilities of two-layer neural nets. In *14th International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 635–638, 1989.

[16] Guido Montúfar, Razvan Pascanu, Kyunghyun Cho, and Yoshua Bengio. On the number of linear regions of deep neural networks. arXiv 1402.1869, February 2014.

[17] János Pach and Gábor Tardos. Separating convex sets by straight lines. *Discrete Mathematics*, 241(1–3):427–433, 2001.

[18] Razvan Pascanu, Guido Montúfar, and Yoshua Bengio. On the number of response regions of deep feed forward networks with piece-wise linear activations. arXiv 1312.6098, December 2013.

[19] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

[20] Ludwig Schläfli. *Theorie der Vielfachen Kontinuität*, volume 38 of *Neue Denkschriften der allgemeinen schweizerischen Gesellschaft für die gesamten Naturwissenschaften*. 1901.

[21] Alessandro Sperduti and Antonia Starita. Speed up learning and network optimization with extended back propagation. *Neural Networks*, 6(3):365–383, 1993.

[22] Helge Tverberg. A separation property of plane convex sets. *Mathematica Scandinavica*, 45:255–260, 1979.

[23] Aaron D. Wyner. Random packings and coverings of the unit n-sphere. *Bell Labs Technical Journal*, 46(9):2111–2118, November 1967.

[24] Guoqiang Peter Zhang. Neural networks for classification: a survey. *IEEE Transactions on Systems, Man, and Cybernetics–Part C: Applications and Reviews*, 30(4):451–462, 2000.