Computers and Chemical Engineering 134 (2020) 106669

Contents lists available at ScienceDirect

Computers and Chemical Engineering

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

Hidden representations in deep neural networks: Part 1. Classification problems

Abhishek Sivaram, Laya Das, Venkat Venkatasubramanian*

Department of Chemical Engineering, Columbia University, New York, NY 10027, USA

ARTICLE INFO

Article history:
Received 27 June 2019
Revised 16 October 2019
Accepted 29 November 2019
Available online 5 December 2019

Keywords:
Deep neural network
Classification
Fault diagnosis
Feature space

ABSTRACT

Deep neural networks have evolved into a powerful tool applicable for a wide range of problems. However, a clear understanding of their internal mechanism has not been developed satisfactorily yet. Factors such as the architecture, number of hidden layers and neurons, and activation function are largely determined in a guess-and-test manner that is reminiscent of alchemy more than of chemistry. In this paper, we attempt to address these concerns systematically using carefully chosen model systems to gain insights for classification problems. We show how wider networks result in several simple patterns identified on the input space, while deeper networks result in more complex patterns. We show also the transformation of input space by each layer and identify the origin of techniques such as transfer learning, weight normalization and early stopping. This paper is an initial step towards a systematic approach to uncover key hidden properties that can be exploited to improve the performance and understanding of deep neural networks.

1. Introduction

In recent years, deep neural networks have witnessed great success and widespread acceptance for solving many challenging problems in different fields. As observed by Venkatasubramanian (2019), this explosive growth is due to three key developments: availability of large amounts of data (i.e., "big data"), availability of powerful and cheap hardware, and advances in user-friendly software environments, all of which can be traced directly to the unexpected success of Moore's Law over the last fifty years. While many ideas such as the back propagation algorithm, convolutional neural nets, recurrent neural nets, Bayesian learning, reinforcement learning, etc., have been around for a couple of decades or more, and therefore are not that "new", what is "new", however, is the ease with which these, and other such techniques, can now be applied in a variety of domains due to the three factors mentioned above.

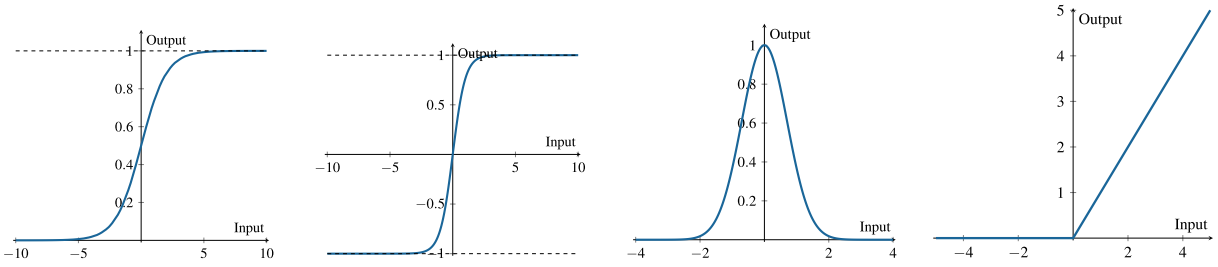
This ease of application of powerful tools has the potential to cause their misuse and abuse, if one is not careful. This concern is similar to what we have witnessed over the years with statistics (Thiese et al., 2015). In this regard, the current state of machine learning is akin to that of alchemy before chemistry and chemical engineering, involving a lot of guess-and-test trials, as noted by Jordan (2018) and Hutson (2018).

The vast majority of recent papers on machine learning, particularly in chemical engineering, is aimed at employing a deep neural network as a tool for solving some application, and not at developing a deeper understanding as to how and why the network derives its problem-solving capability. These papers seem to reflect an attitude that "Well, this particular deep neural network that we specifically crafted after many guess-and-test trials seem to work well enough for this particular application we are interested in, and we don't really care why and how it does so! It does the job. For the next application, we will try something else along these lines, and, hopefully, that will do its job as well!".

Important questions such as what kind of neural network to use, how many hidden layers, how many neurons in each layer, what kind of activation or squashing function to use, how to initialize the weights, and on and on, are largely determined by an Edisonian guest-and-test fashion that reminds one of alchemy more than of chemistry, as noted. A systematic approach to answering these design questions is largely absent in many applications, and such decisions are highly dependent on the expertise (or the lack their of) of the programmer. Despite the seeming practical success of deep neural nets, we find this state of affairs deeply unsatisfactory for at least three reasons. First, it reveals our lack of fundamental understanding of the theory of deep neural networks. Second, this makes the use of neural networks in some engineering applications worrisome due to their lack generalizability and

* Corresponding author.
E-mail addresses: as5397@columbia.edu (A. Sivaram), ld2874@columbia.edu (L. Das), venkat@columbia.edu (V. Venkatasubramanian).

https://doi.org/10.1016/j.compchemeng.2019.106669
0098-1354/



(a) Logistic Function (b) Hyperbolic tangent (c) Gaussian Function (d) Rectified Linear Unit

Fig. 1. Activation functions used in this study.

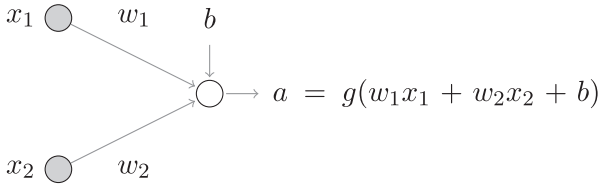


Fig. 2. Architecture for 2 input neurons and 1 activated neuron.

to their potential exposure to adversarial attacks. Such drawbacks are perhaps not very serious in recognizing cats vs dogs, or in recommendation systems (e.g., Yelp, Rotten Tomatoes, etc.), but can be quite important in applications such as fault diagnosis, process safety analysis, where the cost of a mistake could be quite high. Third, there is no causal explanation provided which is needed in many engineering applications.

This paper is written in a tutorial-like manner to highlight these concerns, particularly for the newcomers in machine learning, and to explore and understand the hidden internal representations of deep neural networks to gain some insights for classification problems (the second part of this series explores function approximation problems, to be published shortly). We have chosen deliberately simple examples to reveal key insights about the structure and behavior of deep neural nets, such as the role of depth vs breadth of layers, the different activation or squashing functions and so on. The kinds of examples we have chosen are to be viewed like the "particle in a box" or "simple harmonic oscillator" case studies in quantum mechanics, or like the "ideal gas" system in statistical mechanics. As we know, despite their manifest simplicity, such models have been extraordinarily helpful to understand and appreciate the critical conceptual issues and their interrelationships. They often reveal subtle patterns, and in the case of deep neural networks literally hidden internal representations, which are often hard to perceive in more complex examples.

There have been attempts in the past to explain the functioning of a neural network, using visualization techniques (Yosinski et al., 2015), rule extraction (Setiono and Liu, 1995), randomization approaches to give insights into the importance of nodes in a neural network (Olden and Jackson, 2002), and recently the information bottleneck approach which seems to give the most insight into the functioning of a neural network (Tishby et al., 2000; Tishby and Zaslavsky, 2015; Shwartz-Ziv and Tishby, 2017).

However, from a network design point of view, there are several questions facing a user that remain largely unanswered, rendering the task of designing a network for a certain objective a highly trial-and-error exercise. In addition, there is a general tendency to build deep networks with complicated units such as convolutional filters, recurrent units and inception modules that result in better performance without a clear understanding of the trade-offs involved. It is important to know the minimal neural net architecture for a given problem. Conventionally, an architecture is chosen based on the complexity of the data distribution, number of features, and so on. There is a belief that the deeper the neural network, the better it works - why that is the case is often left as a mystery.

The lack of a clear understanding of the internal mechanism of neural networks partly stems from the intractability of the operations performed by the network. A neural network consists of several nodes arranged in layers that can be connected in different layouts, resulting in intractable operations of the overall network. In this paper, we adopt a systematic approach and make an attempt to understand the operation of the individual components, their arrangements, and finally an entire network. In that sense, our cases represent simple yet rich exemplars (along the lines of the simple harmonic oscillator, particle in a box, and ideal gas molecules, as noted) to gain a deeper understanding. We proceed from an understanding of the individual elements to that of the interactions between them in order to understand the operation of intermediate layers, and then of the complete neural network.

We hope our findings will be of pedagogical value to the new practitioners in the field. Our style of investigation should be of particular interest to chemical engineers who have always prided themselves, historically, as first-principles-based modelers, for whom any "black box" model would be an anathema. This paper is an attempt to inject some transparency into the box, making it more like a "gray box".

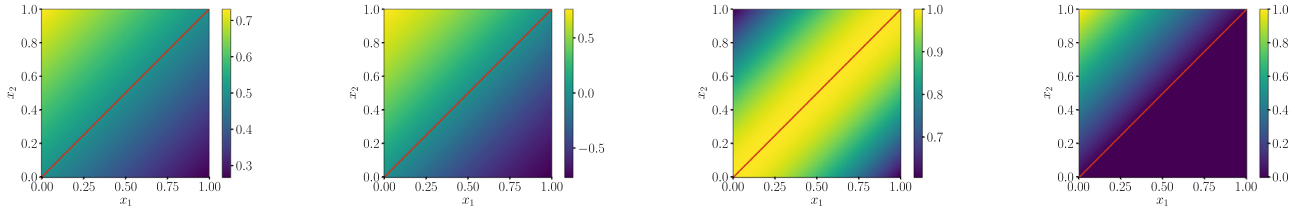
In this paper, we attempt to shed light into the inner workings of the information transformations in a deep neural network. We test simple examples to understand critical issues with the training process. We examine loss function landscapes and show that degeneracy plays a critical role in the parameter space. This in turn results in a degeneracy of observations. Finally, we show what features are estimated by a neural network during the training process for different activation functions. We study three examples which have nonlinear decision boundaries to elucidate this. The generalization of the identified feature space is also studied for a chemical engineering fault diagnosis problem.

2. Mathematical background

2.1. Problem formulation

The dataset in classification tasks comprises (N) examples of input data and their corresponding target classes, represented here as $\mathcal{D} = \{\mathbf{x}^{(i)}, \mathbf{y}^{(i)}\}_{i=1}^{N}$ where $\mathbf{x}^{(i)} \in \mathbb{R}^m$ represents the i^{th} input data and $\mathbf{y}^{(i)}$ is a $K-$ dimensional one-hot vector representing the target vector corresponding to $\mathbf{x}^{(i)}$. The problem of classification is mathematically formulated as an optimization problem wherein the objective (or loss) function is the cross-entropy between the target vector $\mathbf{y}^{(i)}$ and the predicted outputs $\hat{\mathbf{y}}^{(i)}$, defined as:

$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{K} y_j^{(i)} \log \hat{y}_j^{(i)} \tag{1}$$



(a) Logistic Function (σ) (b) Hyperbolic tangent (tanh) (c) Gaussian Function (G) (d) Rectified Linear Unit ($ReLU$)

Fig. 3. Activation of the network shown in Fig. 2 for different regions of the input space with different activation functions. The red line shows the set of points satisfying $w_1x_1 + w_2x_2 + b = 0$ ($w_1 = -1, w_2 = 1, b = 0$). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

The target vector as well as predictions in Eq. (1) represent the probability mass functions (of the target and predicted classes respectively) so that the quantity in Eq. (1) represents the deviation of the predicted outputs (probabilities) from the target, averaged over all examples.

In the above context, a neural network is then a model $\mathcal{N}$ with a predefined architecture $\mathcal{A}$ and a set of parameters Θ that expresses the output as a function of the inputs as:

$$\hat{\mathbf{y}}^{(i)} = \mathcal{N}(\mathbf{x}^{(i)}; \mathcal{A}, \Theta) \tag{2}$$

In this article, we consider the architecture $\mathcal{A}$ to include the design choices made by the user and differentiate them from the parameters Θ that are tuned during training. The architecture $\mathcal{A}$ therefore includes:

1. the organisation of layers, eg., fully connected, convolutional, recurrent
2. the activation function: $g(\cdot)$, eg., logistic function, hyperbolic tangent, rectified linear unit
3. the number of layers L (values ranging from < 10 to $\simeq 150$ in the literature)
4. the number of nodes in each layer n_l, $l = 1, 2, 3, \ldots, L$ (values ranging from ten to few hundreds in the literature)

On the other hand, the parameters Θ include the weights $\mathbf{W}^{[l]}$ and biases $\mathbf{b}^{[l]}$ of each layer, $l = 1, 2, 3, \ldots, L$. It is to be noted that once the architecture $\mathcal{A}$ is defined, the dimensionality of the weights and biases and hence, dimensionality of Θ is fixed.

2.2. Classification with neural networks

The network $\mathcal{N}$ is required to identify (*learn*) the boundaries that can be used to differentiate data belonging to one class from others. These boundaries, in the simplest case, form planes (referred to as *separating hyperplanes*) that divide the input space into multiple regions allowing one to assign different class labels to different regions. However, in several applications these boundaries are extremely curved surfaces, referred to as *separating hypersurfaces*. The nature of the separating boundary results in a linear or nonlinear classification problem, and determines the complexity of a model required to reliably identify the boundaries. It must be noted that in some applications such as identifying lion cubs from cats, it is not possible to manually identify distinct hypersurfaces in the input space, and it becomes necessary to extract certain *features* relevant to the classification problem. Therefore, achieving classification with neural networks, which entails identifying appropriate separating boundaries from the data typically involves *feature extraction* followed by *classification*. While the hidden layers are used to extract relevant features from the data, the final layer is used to identify the separating surfaces necessary for classification.

Let us consider a multi-layer network $\mathcal{N}$ with L layers and the l^{th} layer consisting of n_l nodes and an activation function $g^{[l]}(\cdot)$. Let $\mathbf{z}^{[l]}$, $\mathbf{W}^{[l]}$, $\mathbf{b}^{[l]}$, $g^{[l]}(\cdot)$ and $\mathbf{a}^{[l]}$ represent the input, weights, biases, activation function and output of the lth layer respectively. The output $\hat{\mathbf{y}}$ of the network can then be obtained through a progression of layer-wise processing of the input $\mathbf{x}$ as:

$$\begin{aligned}
\mathbf{a}^{[1]} &= g^{[1]}\left(\mathbf{z}^{[1]}\right) = g^{[1]}\left(\mathbf{W}^{[1]}\mathbf{x} + \mathbf{b}^{[1]}\right) \\
\mathbf{a}^{[l]} &= g^{[l]}\left(\mathbf{z}^{[l]}\right) = g^{[l]}\left(\mathbf{W}^{[l]}\mathbf{a}^{[l-1]} + \mathbf{b}^{[l]}\right) \quad \forall l \in \{2, \ldots, L-1\} \\
\hat{\mathbf{y}} &= g^{[L]}\left(\mathbf{W}^{[L]}\mathbf{a}^{[L-1]} + \mathbf{b}^{[L]}\right)
\end{aligned} \tag{3}$$

The output $\hat{\mathbf{y}}$ is therefore a function of the inputs $\mathbf{x}$, the parameters $\mathbf{W}^{[l]}$, $\mathbf{b}^{[l]}$, $l = 1, 2, 3, \ldots, L$ and the activation function $g^{[l]}(\cdot)$, $l = 1, 2, 3, \ldots, L$. It must be noted here that while the expressive power of a neural network lies in the architecture chosen for a particular task, its actual performance depends on the architecture as well as the values of the parameters. The host of algorithms developed for training neural networks attempt to tune the parameters Θ of the network $\mathcal{N}$ so as to minimize the loss. In the following, we discuss these factors and highlight the different scenarios considered in this work.

Table 1
Activation functions and their ranges.

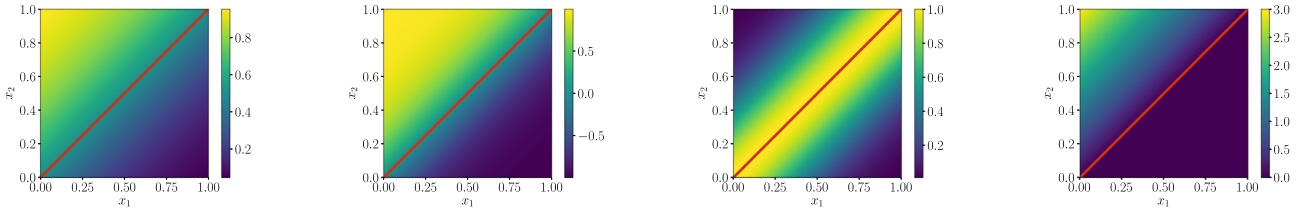
Activation Function (g)	Expression	Range
Logistic Function (σ)	$\frac{1}{1+\exp(-z)}$	[0,1]
Hyperbolic tangent ($tanh$)	$\frac{\exp(z)-\exp(-z)}{\exp(z)+\exp(-z)}$	$[-1, 1]$
Gaussian Function (G)	$\exp(-z^2)$	[0,1]
Rectified Linear Unit ($ReLU$)	$\max(0, z)$	$\mathbb{R}^+$

The choice of the number of nodes in a layer (width), number of hidden layers (depth), type of connection (fully connected, convolutional, recurrent) and the activation function (denoted as $g(\cdot)$) used in each layer together determine the expressive power of a neural network. In this work, we consider multiple combinations of width and depth for fully connected networks to identify the influence of these factors on the performance of the network. In addition, we consider the logistic function σ, hyperbolic tangent function *tanh*, Gaussian function G and the rectified linear unit *ReLU* as different activation functions used in the hidden layers in the networks. Table 1 lists the definitions and ranges of the different activation functions used in this study. The different activation functions are also shown in Fig. 1.

In addition to the above, the final layer of the network consists of the *softmax* activation function, which is used to produce the probability of each class for the input, and is defined for a layer with K nodes (corresponding to a K - class problem) as follows:

$$P(y = j) \triangleq \hat{y}_j = \frac{\exp(z_j)}{\sum_{k=1}^{K} \exp(z_k)} \qquad \forall j \in \{1, \ldots, K\} \tag{4}$$

In the following, we present a systematic approach to understanding the internal mechanism of a neural network. We begin with a simple network that is the equivalent of a single-cellular or-



(a) Logistic Function (σ) (b) Hyperbolic tangent (tanh) (c) Gaussian Function (G) (d) Rectified Linear Unit ($ReLU$)

Fig. 4. Activation of the network shown in Fig. 2 for different regions of the input space with different activation functions. The red line shows the set of points satisfying $w_1x_1 + w_2x_2 + b = 0$ ($w_1 = -3, w_2 = 3, b = 0$). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

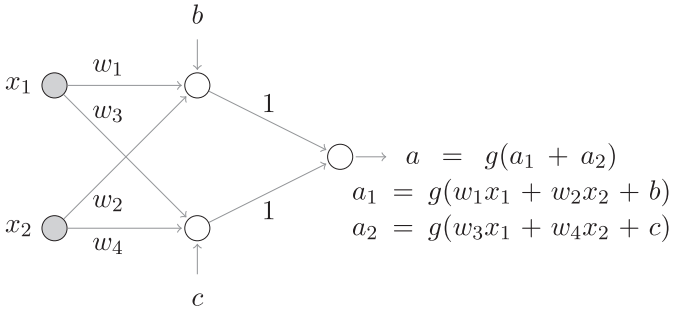


Fig. 5. Architecture for 2 input neurons and 1 activated neuron, with 1 hidden layer.

ganism, or a particle in a box, that lends itself to a dissection-like study providing insights about larger networks with complicated architectures. We then investigate the impact of different components of the architecture $\mathcal{A}$ listed earlier on the performance of a neural network.

3. Peeking under the hood of a deep neural network

A neural network is made up of multiple neurons arranged in a layer that are connected to neurons in other layers. Each node in the lth hidden layer of a network activates a portion of the space spanned by the nodes of $l-1^{\text{th}}$ layer, according to the parameters $\mathbf{W}^{[l]}$, $\mathbf{b}^{[l]}$ and the activation function $g^{[l]}(\cdot)$. As discussed earlier, a network performs *feature extraction* from the input data with the hidden layers, which are then used for performing *classification* at the final layer, which are analysed in the same order in the following.

3.1. Feature extraction: Node-specific selective activation of the input space

Let us consider a tiny neural network that consists of two input nodes and one output node as shown in Fig. 2. The output activation of the network can be expressed as:

$$a = g(w_1x_1 + w_2x_2 + b)$$

For this network, let us define a node-specific *characteristic equation* that determines the activation of the input space as:

$$w_1x_1 + w_2x_2 + b = 0 \tag{5}$$

The above equation describes a line in the 2 dimensional input space that acts as the reference for selectively activating different regions of the space.

Fig. 3 shows the output of the network for different activation functions $g(\cdot)$ used in the output node. In each plot, the set of points satisfying the characteristic equation (with $w_1 = -1$, $w_2 = 1$ and $b = 0$) is shown as a red line, while the value of the output is shown as a gradient of colour with yellow representing the highest value and deep blue representing the lowest value. It is noteworthy that each point (X_1, X_2) in the input space is activated according to the scaled, rotated and translated value $d(X_1, X_2) = w_1X_1 + w_2X_2 + b$, which is also equal to the scaled directional distance of the point (X_1, X_2) from the line segment representing the characteristic equation, scaled by the norm of $\mathbf{w} = [w_1, w_2]$. Henceforth d shall be refered to as the distance.

Different types of activation functions however result in different segments of the input space being activated as shown in Fig. 3. It can be observed from Fig. 3 that while σ results in the line $w_1x_1 + w_2x_2 + b = 0$ being assigned a value of 0.5, *tanh* suppresses the same to zero, activating other points according to their distance from the line. On the other hand, *ReLU* suppresses all inputs that satisfy $w_1x_2 + w_2x_2 + b \leq 0$ while activating others by an amount equal to their magnitude. Finally, the Gaussian activation results in the line $w_1x_1 + w_2x_2 + b = 0$ being activated with the highest magnitude, with the activation decaying as a function of the distance of the data points from the line.

As mentioned earlier, in addition to the form of the activation function, the magnitude of activation of an input data point (X_1, X_2) also depends on its distance from the line representing the characteristic equation, equal to $w_1X_1 + w_2X_2 + b$ which is tuned at the time of training. This has two implications on the training of neural networks. *First*, the parameters w_1, w_2 and b can be tuned over a continuum of values resulting in different lines in the input space and thus different levels of activation. *Second*, it must be noted that the set of points that satisfy $w_1x_1 + w_2x_2 + b = 0$ also satisfy the equation $\alpha w_1x_1 + \alpha w_2x_2 + \alpha b = 0$ for all non-zero values of α. As a result, the magnitude of activation of any given input data point can also be tuned over a continuum of values by adjusting the value of α. Furthermore, the values assigned to data points on different sides of the line can be interchanged by accommodating negative values of α. Fig. 4 shows the output of the network for different activation functions $g(\cdot)$ with set of points satisfying the characteristic equation (with $w_1 = -3$, $w_2 = 3$ and $b = 0$) is shown as a red line and a similar gradient of colour representing different levels of activation. A comparison of Figs. 3 and 4 reveals that the latter results in a much greater gradient and/or magnitude of activation resulting from the larger value of $||\mathbf{w}||$ for the network. This demonstrates the impact of α on the activation levels for each node, and the extent of nonlinearity achieved with the same activation function.

We discussed the behaviour of a single node in a neural network as a function of the parameters $\mathbf{w}$, b and the activation function $g(\cdot)$ and identified two different methods of adjusting the parameters to selectively extract features from the input space. In the following, we add more resources to the tiny network and study the impact of depth and width of the resulting network.

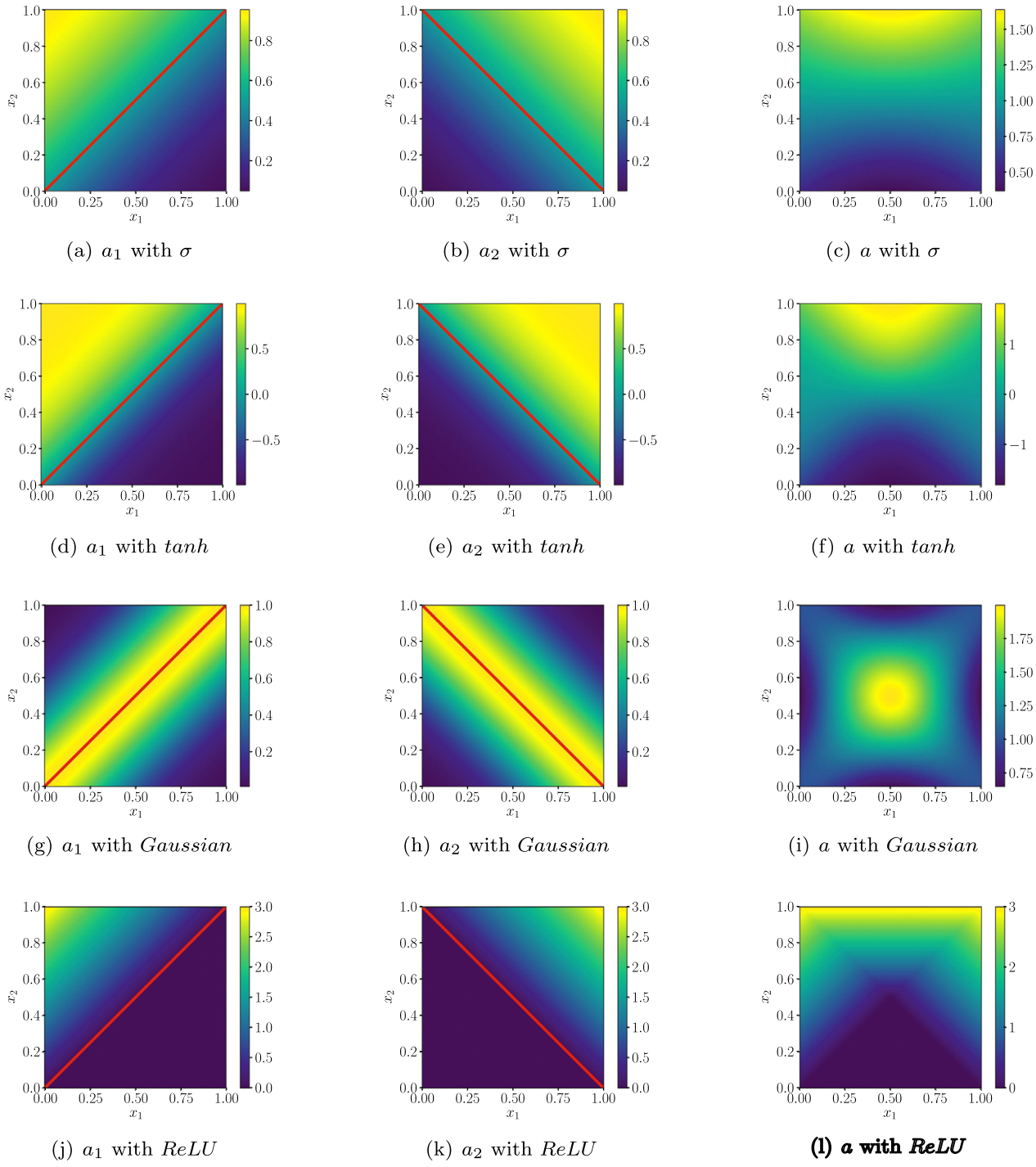


Fig. 6. Input space activation for each node in the network architecture as shown in Fig. 5. The values of the weights and biases are chosen to be $w_1, w_2, b = -3, 3, 0$, $w_3, w_4, c = 3, 3, -3$.

3.2. Wider vs deeper networks: Complexity of features

Let us add another neuron at the output layer of the tiny network and combine the outputs of the nodes to generate the final output of the new network, as shown in Fig. 5. The output of this network can now be expressed as:

$$a = g(a_1 + a_2) = g\big(g(w_1x_1 + w_2x_2 + b) + g(w_3x_1 + w_4x_2 + c)\big)$$

It is now possible to study the activation pattern in each of the three nodes, as shown in Fig. 6 for different activation functions. It can be observed from Fig. 6 that the two neurons in the hidden layer activate different segments of the input space according to their characteristic equations: $w_1x_1 + w_2x_2 + b = 0$ and $w_3x_1 + w_4x_2 + c = 0$. These activations are then combined to produce the final output that activates different patches in the input space. It must be noted here that while the nodes in the first layer selectively activated the input space according to a characteristic equation representing a line, the node in the final layer activates the input space according to a characteristic equation representing a curve, which is obtained through combination and composition of the different activations.

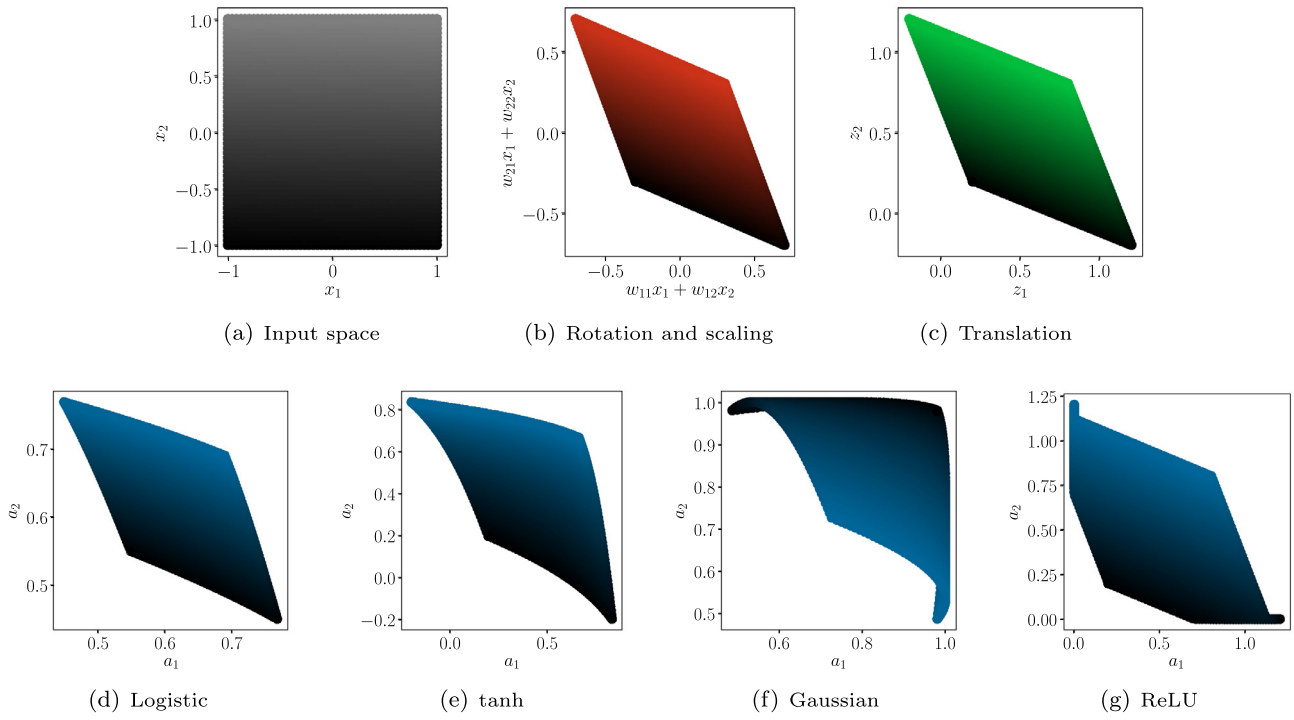


Fig. 7. Example transformations of the input space to the feature space. The Input space (a) is first rotated and stretched (b), then translated to give the **z** vector (c). Note how all the transformations are affine so far. The final nonlinear operation comes from the squishing action based on different activation function on the transformed space to give the final feature space representation (d),(e),(f) and (g). The dark regions in the input space are mapped to dark regions in the subsequent feature space.

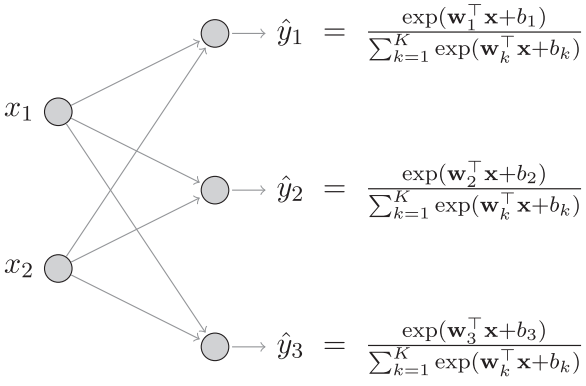


Fig. 8. Sample architecture for linearly separable problems.

It can therefore be observed that with the addition of more number of nodes to the tiny network, it is imparted with the ability to generate more number of selective activations, which are *simple* and characterized by a *straight line* in the input space. However, with the addition of more layers to the network, the network can create *complex* selective activations that are characterized by *arbitrary curves* in the input space. These straight lines and curves are tuned at the time of training such that each node in each layer selectively extract a portion of the input space that is relevant to the classification task.

A careful examination of Figs. 4 and 6 reveals that in a network $\mathcal{N}$ with L layers, the nodes in the first layer will always result in activations representing straight lines in the input space, and as one builds a deeper network, these lines are transformed into curved geometric objects. In other words, the characteristic equation of nodes in the first layer represent straight lines, which become progressively nonlinear as one goes deep into the network - irrespective of the task performed by the network. Now, if two networks are trained to perform different but related tasks, it is likely for the geometric objects identified by the networks to have similar descriptions. As a result of this generic increase in the complexity of the characteristic equation and dependence of the characteristic equation on the final task, it is possible to make use of pre-trained networks as initializations and to train new networks for achieving related tasks with less computational time, a process referred to as *transfer learning* in the literature (Pan and Yang, 2009; Taylor and Stone, 2009; Weiss et al., 2016).

For example, consider two networks $\mathcal{N}_1$ and $\mathcal{N}_2$ that are trained to achieve different, but related classification tasks, eg., to differentiate cats from other animals ($\mathcal{N}_1$) and differentiate cats from dogs ($\mathcal{N}_2$). Both these networks, at the time of training, learn a set of linear and curved geometric objects in the input space that can be used to achieve their corresponding classification tasks. However, since both the networks attempt to distinguish cats from other objects (all other animals for $\mathcal{N}_1$ and dogs for $\mathcal{N}_2$), it stands to reason that they identify characteristics explicit to cats in relation to the other objects. As a result, several of the geometric objects identified by $\mathcal{N}_1$ (to distinguish cats from other animals) are likely to be similar (if not same) to those identified by $\mathcal{N}_2$ (to distinguish cats from dogs). A pre-trained network that distinguishes cats from other animals and has already identified features relevant to cats can therefore act as a better initialization for a network that is needed to distinguish cats from dogs, resulting in requirement of tuning of only a few parameters, and hence faster convergence. This process of using a pre-trained network $\mathcal{N}_1$ as the initialization for achieving a different but related task is referred to as transfer learning. This technique has been used for classification of abnormalities in brain from magnetic resonance images using ResNet34 as the pre-trained network (Talo et al., 2019), computer aided detection problems (Shin et al., 2016) with CifarNet, AlexNet and GoogLeNet networks and for detection of cracks and distress on pavements with the VGG-16 network (Gopalakrishnan et al., 2017).

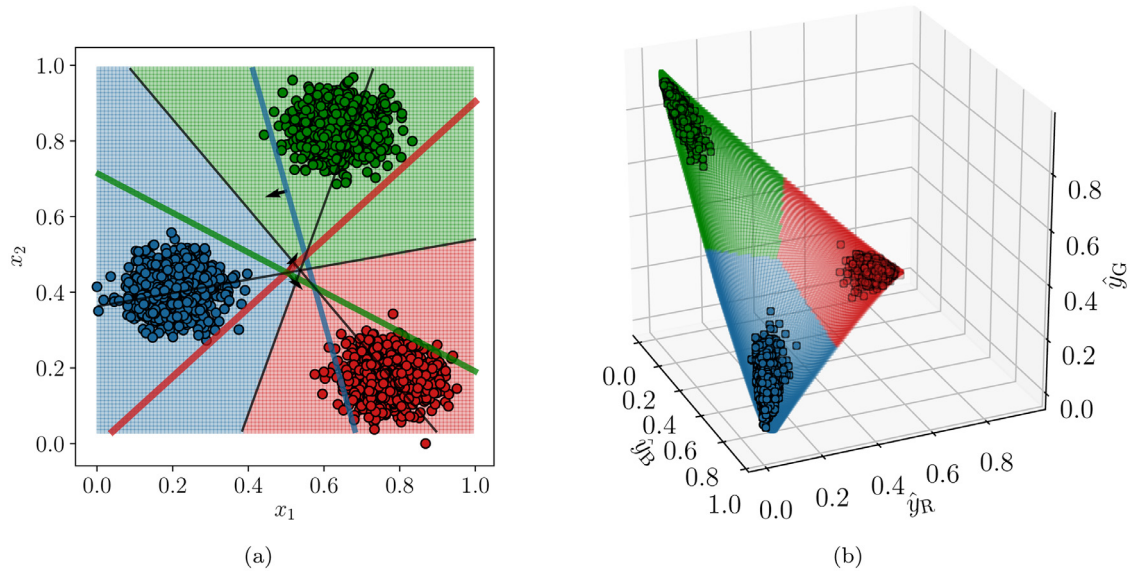


Fig. 9. (a) Final estimated segregation on the input space. Thickness indicates the magnitude of $\mathbf{w}_j$ for class j. The black lines are constructed using the boundary equation–Eq. (7). (b) Identified probabilities of each point on the input space.

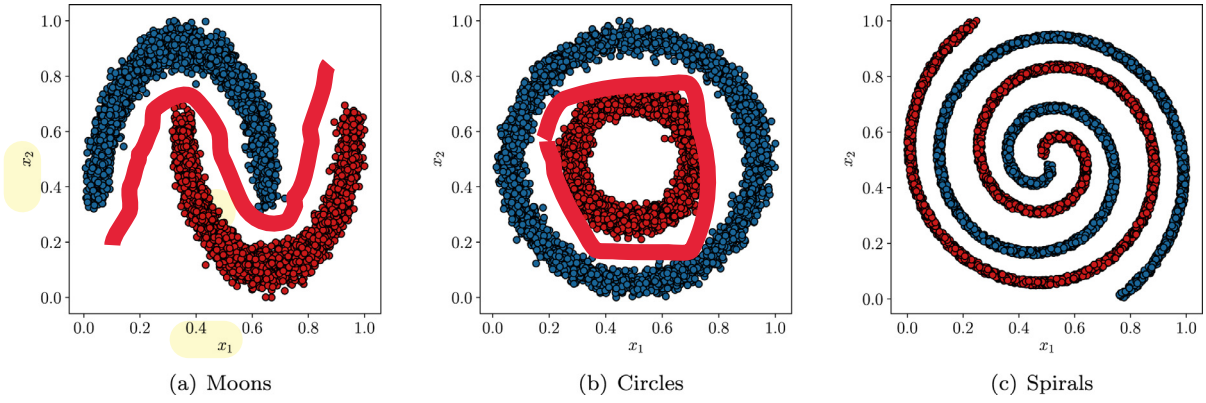


Fig. 10. Example datasets for studying the operation of neural networks.

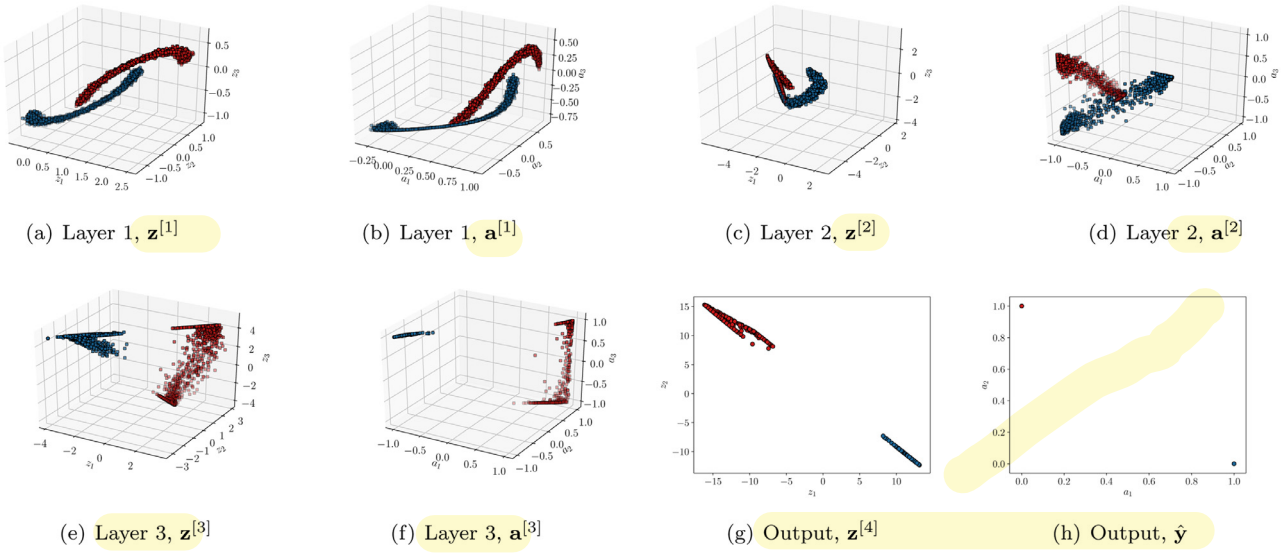
In this section, we identified the impact of width and depth of a neural network on the characteristic equations of nodes in different layers of the network. Specifically, while wider networks exhibit more number of simple characteristic equations and hence activations, deeper networks exhibit more complex equations describing complex structures in the input space. We also identified a generic pattern of the complexity of the characteristic equation, as one builds a deep neural network, and how this is exploited in the form of transfer learning in the literature. In the following, we focus our attention on the final layer of the classifier that takes the features of the pre-final layer and identifies separating boundaries in the feature space.

3.3. From features to feature spaces

We have seen that intermediate nodes of the neural network results in selective activation of input space and creation of complex patterns for classification. Independent activation of each of these nodes in the intermediate layers, results in transformation of the input space to generate intermediate *feature spaces* on which subsequent layers operate. The different operations performed on the input space, and the resultant feature spaces for different activation functions are shown in Fig. 7. For this illustration, $\mathbf{W} = \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{bmatrix} = \begin{bmatrix} 0.5 & -0.2 \\ -0.2 & 0.5 \end{bmatrix}$, $\mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix}$. The input space – a flat plane – is first *rotated* and *scaled* according to weights in the weight matrix $\mathbf{W}$. This is followed by *translation* of the rotated and scaled space according to the bias $\mathbf{b}$. Finally, the translated space is *transformed* according to the nonlinear activation function $g(\cdot)$ to produce a curved surface from the flat input space. Fig. 7 also compares the transformation achieved with different activation functions. In a neural network with multiple layers, the feature space of one layer acts as the input space for the successive layer. In such a network, the input space gets progressively transformed at every successive layer to generate intermediate feature spaces of which the final one is used for classification.

3.4. The final layer: Separating hyperplanes for classification

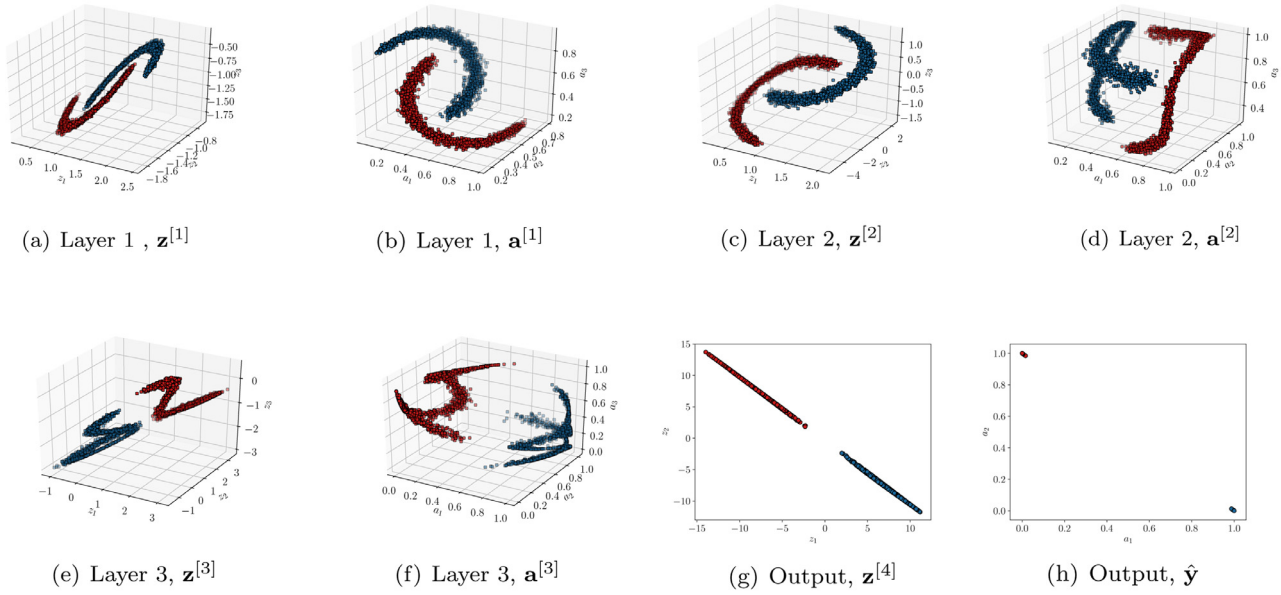
The final layer of a neural network classifier solving a K-class classification problem typically employs the *softmax* activation that produces at the output, the probabilities of the input belonging to different classes. It must be noted that the *softmax* activation is a linear classifier, implying that it classifies different classes in *its* input space by identifying *hyperplanes* that (potentially) minimise the objective function. The number of nodes and activation function of the final layer of a neural network are therefore fixed, with only the parameters of this layer playing a crucial in identifying the separating boundaries. In order to study the properties of the weights of this layer, we consider a dataset with three classes that



(a) Layer 1, $\mathbf{z}^{[1]}$ (b) Layer 1, $\mathbf{a}^{[1]}$ (c) Layer 2, $\mathbf{z}^{[2]}$ (d) Layer 2, $\mathbf{a}^{[2]}$

(e) Layer 3, $\mathbf{z}^{[3]}$ (f) Layer 3, $\mathbf{a}^{[3]}$ (g) Output, $\mathbf{z}^{[4]}$ (h) Output, $\hat{\mathbf{y}}$

Fig. 11. Internal internal representation of network with $n = 3, H = 3$ and *tanh* as the activation function trained on the moons dataset.



(a) Layer 1 , $\mathbf{z}^{[1]}$ (b) Layer 1, $\mathbf{a}^{[1]}$ (c) Layer 2, $\mathbf{z}^{[2]}$ (d) Layer 2, $\mathbf{a}^{[2]}$

(e) Layer 3, $\mathbf{z}^{[3]}$ (f) Layer 3, $\mathbf{a}^{[3]}$ (g) Output, $\mathbf{z}^{[4]}$ (h) Output, $\hat{\mathbf{y}}$

Fig. 12. Internal internal representation of network with $n = 3, H = 3$ and *Gaussian* activation function trained on the moons dataset.

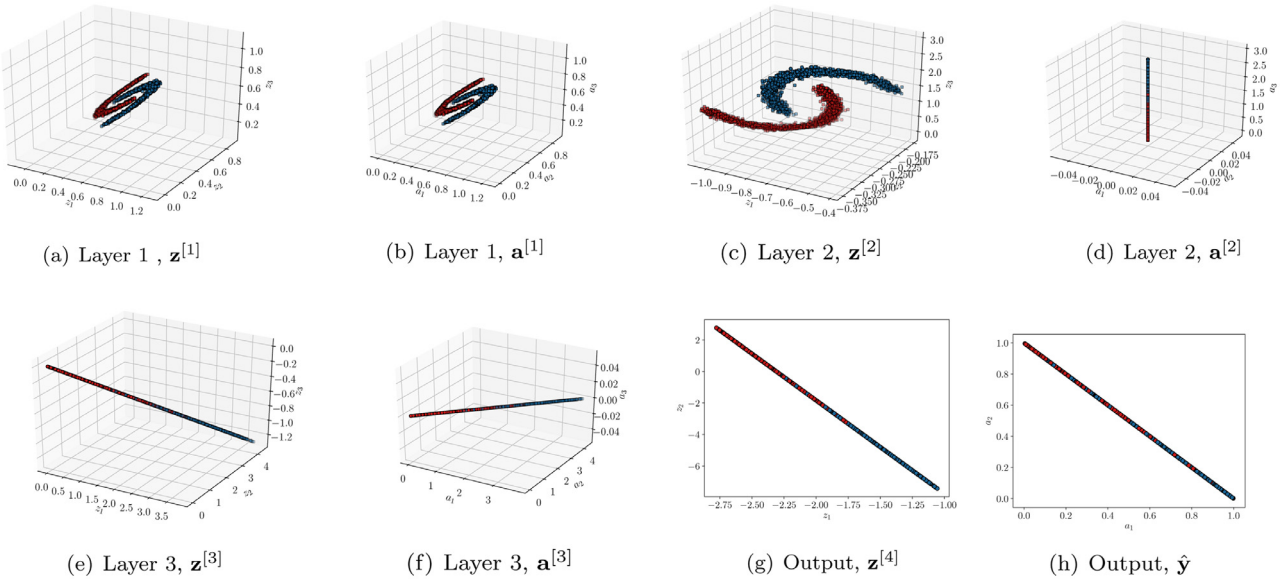
are linearly separable, and that are classified with a *softmax* classifier as shown in Fig. 8.

The *softmax* classifier shown in Fig. 8 (with $K = 3$) can be expressed as:

$$\hat{y}_j^{(i)} = \frac{\exp\left(\mathbf{w}_j^\top \mathbf{x}^{(i)} + b_j\right)}{\sum_{k=1}^{3} \exp\left(\mathbf{w}_k^\top \mathbf{x}^{(i)} + b_k\right)} \tag{6}$$

The quantity $\mathbf{w}_j^\top \mathbf{x} + b_j$ represents the distance of a point $\mathbf{x}$ on the input space from the line $\mathbf{w}_j^\top \mathbf{x} + b_j = 0$, i.e., the characteristic equation of the jth output node. It can thus be observed that much like the feature extraction in the hidden layers, the classification performed in the final layer with a *softmax* activation also makes use of the distance of data points on the input space from the characteristic equation.

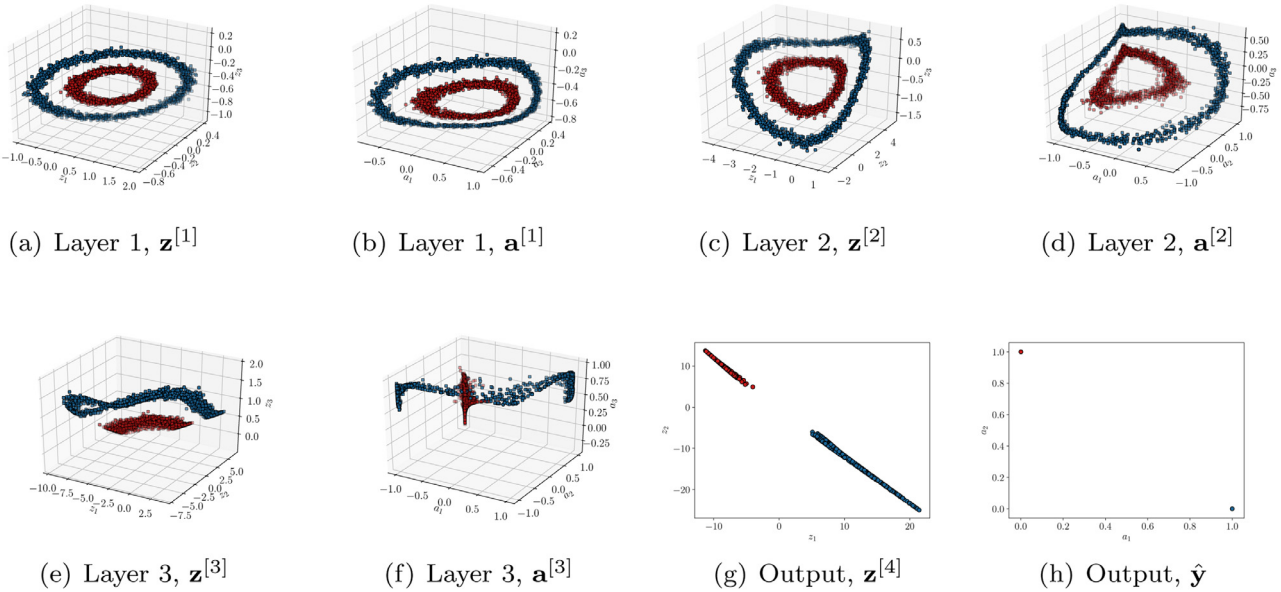
Fig. 9a shows the data points belonging to the three classes highlighted in different colours, the lines representing the characteristic equations of each node in the *softmax* layer and the decision boundaries for the three classes identified by the network. The width of the lines representing the characteristic equations in Fig. 9a are proportional to the norm of the corresponding weight vector consisting of two weights per class. The membership of each point on the input space to each class is characterized by its distance from the class's hyperplane. The arrows point in the direction of positive distance/membership. The greater the membership of a point to a given hyperplane, the higher its probability of belonging to that class. For example, in Fig. 9a the red line represents the equation $3x_1 - 3.3x_2 - 0.01 = 0$ and the arrow represents the region of the input space with positive distance from the red line. The region highlighted in red has the greatest membership with respect to the red line, in comparison to the other lines. Using Eq. (6), we get the final probability vectors for all the points on the input space (Fig. 9b). Notice how, due to the constraint that $\sum_{k=1}^{3} \hat{y}_k = 1$, the final probability vector lies on the diagonal plane. In fact, for K-dimensional classification problem, when one uses a *softmax* activation in the final layer, the resultant vector will lie on a K-dimensional diagonal *hyperplane*.



(a) Layer 1 , $\mathbf{z}^{[1]}$ (b) Layer 1, $\mathbf{a}^{[1]}$ (c) Layer 2, $\mathbf{z}^{[2]}$ (d) Layer 2, $\mathbf{a}^{[2]}$

(e) Layer 3, $\mathbf{z}^{[3]}$ (f) Layer 3, $\mathbf{a}^{[3]}$ (g) Output, $\mathbf{z}^{[4]}$ (h) Output, $\hat{\mathbf{y}}$

Fig. 13. Internal representation of the network with $n = 3, H = 3$ and *ReLU* as the activation function trained on the moons dataset.



(a) Layer 1, $\mathbf{z}^{[1]}$ (b) Layer 1, $\mathbf{a}^{[1]}$ (c) Layer 2, $\mathbf{z}^{[2]}$ (d) Layer 2, $\mathbf{a}^{[2]}$

(e) Layer 3, $\mathbf{z}^{[3]}$ (f) Layer 3, $\mathbf{a}^{[3]}$ (g) Output, $\mathbf{z}^{[4]}$ (h) Output, $\hat{\mathbf{y}}$

Fig. 14. Internal representation of network with $n = 3, H = 3$ and *tanh* as activation function trained on circles dataset.

From the discussion so far, we can say that if a point $\mathbf{x}$ in the input space, belongs to cluster c, then,

$$\mathbf{w}_c^\top \mathbf{x} + b_c > \mathbf{w}_k^\top \mathbf{x} + b_k \quad k \neq c$$

It follows that the decision boundary between any two classes i and j can be identified using the hyperplanes of said classes, as the set of points $\mathbf{x}$ where the memberships are equal, i.e.,

$$\mathbf{w}_i^\top \mathbf{x} + b_i = \mathbf{w}_j^\top \mathbf{x} + b_j$$

$$\left(\mathbf{w}_i - \mathbf{w}_j\right)^\top \mathbf{x} + \left(b_i - b_j\right) = 0 \tag{7}$$

As a result, the final layer exhibits severe degeneracy of parameters, which are described in the following.

3.5. Degeneracy of parameters using softmax activation

Translation degeneracy: Consider a set of parameters (**W, b**) that can accurately classify the data. Consider vector $\mathbf{w}_0$ and scalar b_0, $\mathbf{w}_0 \in \mathbb{R}^m, b_0 \in \mathbb{R}$. Then the translated set of parameters $(\mathbf{W} + \mathbf{W}_0, \mathbf{b} + \mathbf{b}_0)$, where $\mathbf{W} + \mathbf{W}_0 = [\mathbf{w}_1 + \mathbf{w}_0, \quad \mathbf{w}_2 + \mathbf{w}_0, \quad \cdots, \quad \mathbf{w}_K + \mathbf{w}_0]^\top \in \mathbb{R}^{K \times m}$ and $\mathbf{b} + \mathbf{b}_0 = [b_1 + b_0, \quad b_2 + b_0, \quad \cdots, \quad b_K + b_0]^\top$, will also classify the data with the same accuracy.

Proof. Let a data point $\mathbf{x}^{(i)}$ belong to class c_i. Then, $\forall\, i \in \{1, \ldots, N\}$ that are correctly classified by the classifier,

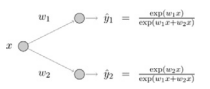
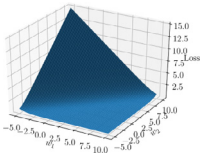
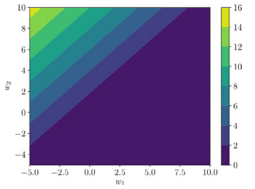
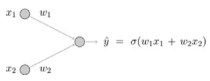
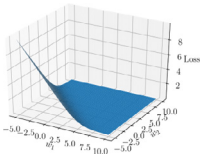
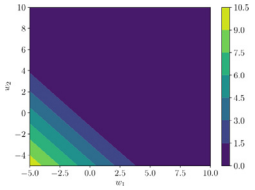
$$\mathbf{w}_{c_i}^\top \mathbf{x}^{(i)} + b_{c_i} > \mathbf{w}_k^\top \mathbf{x}^{(i)} + b_k, \quad k \neq c_i$$

$$\Rightarrow \mathbf{w}_{c_i}^\top \mathbf{x}^{(i)} + b_{c_i} + \mathbf{w}_0^\top \mathbf{x}^{(i)} + b_0 > \mathbf{w}_k^\top \mathbf{x}^{(i)} + b_k + \mathbf{w}_0^\top \mathbf{x}^{(i)} + b_0, \quad k \neq c_i$$

$$\Rightarrow (\mathbf{w}_{c_i} + \mathbf{w}_0)^\top \mathbf{x}^{(i)} + (b_{c_i} + b_0) > (\mathbf{w}_k + \mathbf{w}_0)^\top \mathbf{x}^{(i)} + (b_k + b_0), \quad k \neq c_i$$

The decision boundary estimated using Eq. (7), remains the same. In the event of $\mathbf{w}_0$ being a zero vector, one is essentially translating each of the class hyperplanes by the same magnitude, and the overall effect of this is nullified. In fact, it can be seen that the translated weights and biases give the same set of output probabilities as the untranslated weights and biases, and hence, loss

Table 2
The nature of the Loss function exhibiting degeneracy of parameters .

Label	Architecture	Loss landscape	2 D projection Loss landscape
a			
b			

value.

$$\begin{aligned}\hat{y}_j(\mathbf{W}+\mathbf{W}_0,\mathbf{b}+\mathbf{b}_0) &= \frac{\exp\left((\mathbf{w}_j+\mathbf{w}_0)^\top\mathbf{x}+(b_j+b_0)\right)}{\sum_{k=1}^{K}\exp\left((\mathbf{w}_k+\mathbf{w}_0)^\top\mathbf{x}+(b_k+b_0)\right)}\\ &= \frac{\exp\left(\mathbf{w}_j^\top\mathbf{x}^{(i)}+b_j\right)\exp\left(\mathbf{w}_0^\top\mathbf{x}+b_0\right)}{\sum_{k=1}^{K}\exp\left(\mathbf{w}_k^\top\mathbf{x}^{(i)}+b_k\right)\exp\left(\mathbf{w}_0^\top\mathbf{x}+b_0\right)}\\ &= \frac{\exp\left(\mathbf{w}_j^\top\mathbf{x}+b_j\right)}{\sum_{k=1}^{K}\exp\left(\mathbf{w}_k^\top\mathbf{x}+b_k\right)}\\ &= \hat{y}_j(\mathbf{W},\mathbf{b})\end{aligned}$$

□

Multiplicative degeneracy: If a set of parameters (**W, b**) can accurately classify the data, the set of parameters ($\alpha\mathbf{W}$, $\alpha\mathbf{b}$), $\alpha > 0$ will also classify the data with the same accuracy.

Proof. $\forall i \in \{1,\ldots,N\}$ that are correctly classified by the classifier,

$$\begin{aligned}&\mathbf{w}_{c_i}^\top\mathbf{x}^{(i)}+b_{c_i} > \mathbf{w}_k^\top\mathbf{x}^{(i)}+b_k, \quad k\neq c_i\\ \Rightarrow\ &\alpha\mathbf{w}_{c_i}^\top\mathbf{x}^{(i)}+\alpha b_{c_i} > \alpha\mathbf{w}_k^\top\mathbf{x}^{(i)}+\alpha b_k, \quad k\neq c_i, \alpha>0\end{aligned}$$

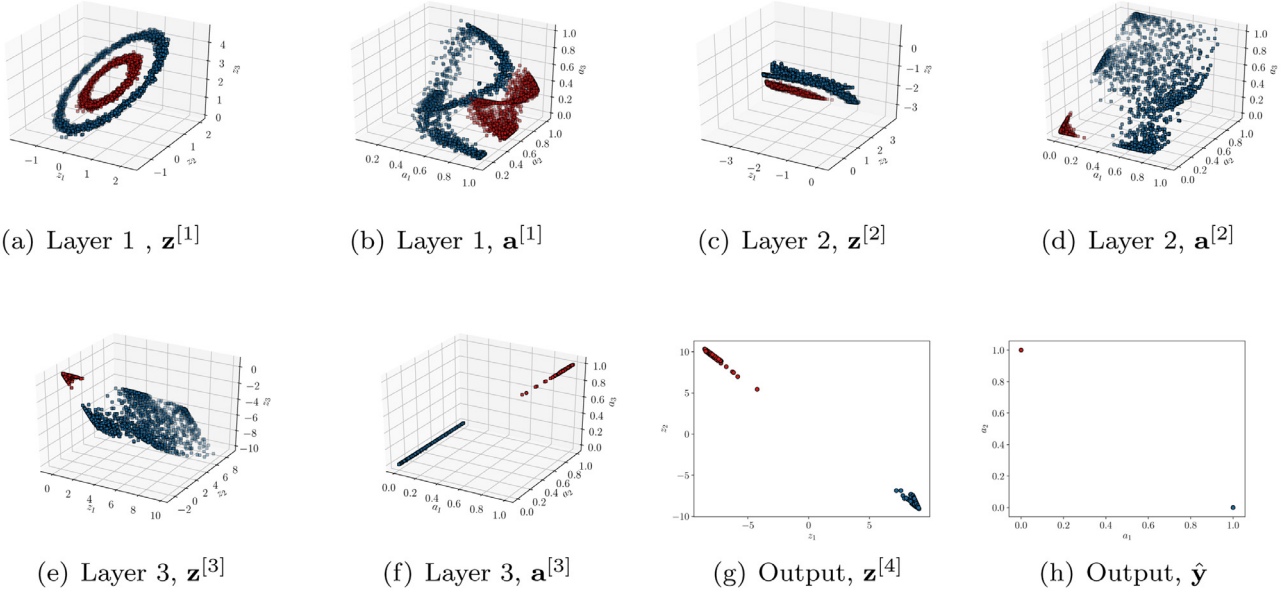
By a similar argument for the examples that are incorrectly classified, it follows that the accuracy of the classifier remains the same. □

Additive degeneracy: If a set of parameters (**W, b**) and (**W′, b′**) can accurately classify the data, the set of parameters (**W** + **W′, b** + **b′**), will also classify the data with the same accuracy. $\forall i \in \{1,\ldots,N\}$ that are correctly classified by both the classifiers,

$$\begin{aligned}&\mathbf{w}_{c_i}^\top\mathbf{x}^{(i)}+b_{c_i} > \mathbf{w}_k^\top\mathbf{x}^{(i)}+b_k, \quad k\neq c_i\\ &\mathbf{w}_{c_i}'^\top\mathbf{x}^{(i)}+b'_{c_i} > \mathbf{w}_k'^\top\mathbf{x}^{(i)}+b'_k, \quad k\neq c_i\\ \Rightarrow\ &(\mathbf{w}_{c_i}+\mathbf{w}'_{c_i})^\top\mathbf{x}^{(i)}+(b_{c_i}+b'_{c_i}) > (\mathbf{w}_k+\mathbf{w}'_k)^\top\mathbf{x}^{(i)}+(b_k+b'_k), \quad k\neq c_i\end{aligned}$$

Superposition degeneracy: Combining the additive and multiplicative degeneracy, we can see that if a set of parameters (**W, b**) and (**W′, b′**) can accurately classify the data, the set of parameters ($\alpha\mathbf{W}+\beta\mathbf{W}'$, $\alpha\mathbf{b}+\beta\mathbf{b}'$) where α, $\beta > 0$, will also classify the data with the same accuracy.

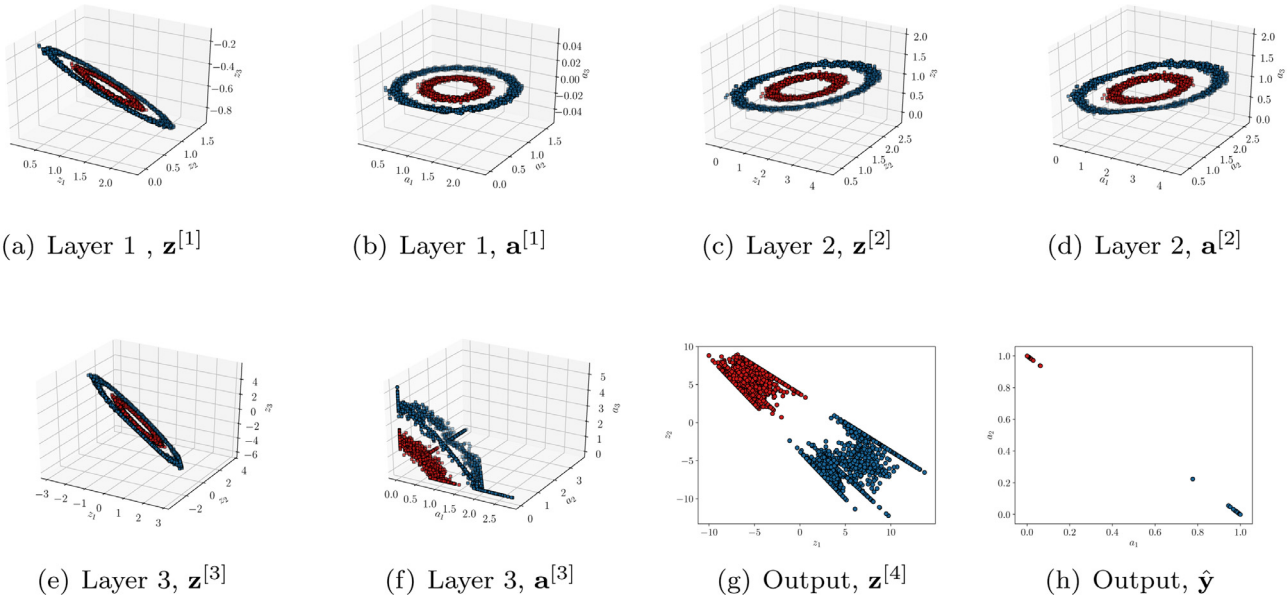
Table 2 shows the impact of degeneracy of parameters as a function of the weights for two different architectures. It can be observed from Table 2 that the loss decreases sharply for only a limited range of weights and remains relatively flat for majority



(a) Layer 1 , $\mathbf{z}^{[1]}$ (b) Layer 1, $\mathbf{a}^{[1]}$ (c) Layer 2, $\mathbf{z}^{[2]}$ (d) Layer 2, $\mathbf{a}^{[2]}$

(e) Layer 3, $\mathbf{z}^{[3]}$ (f) Layer 3, $\mathbf{a}^{[3]}$ (g) Output, $\mathbf{z}^{[4]}$ (h) Output, $\hat{\mathbf{y}}$

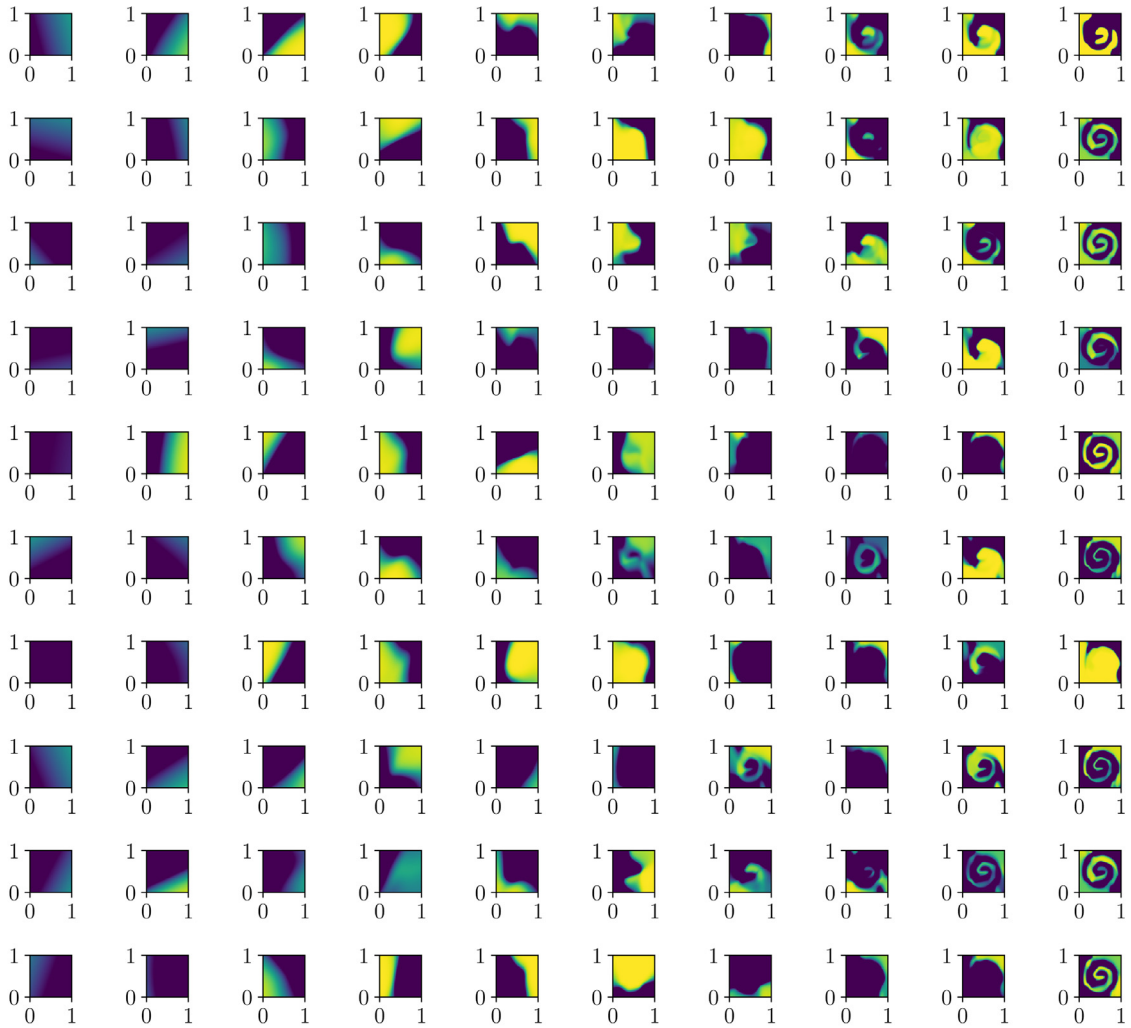
Fig. 15. Internal representation of network with $n = 3, H = 3$ and *Gaussian* activation function trained on the circles dataset.



(a) Layer 1 , $\mathbf{z}^{[1]}$ (b) Layer 1, $\mathbf{a}^{[1]}$ (c) Layer 2, $\mathbf{z}^{[2]}$ (d) Layer 2, $\mathbf{a}^{[2]}$

(e) Layer 3, $\mathbf{z}^{[3]}$ (f) Layer 3, $\mathbf{a}^{[3]}$ (g) Output, $\mathbf{z}^{[4]}$ (h) Output, $\hat{\mathbf{y}}$

Fig. 16. Internal representation of network with $n = 3, H = 3$ and *ReLU* as the activation function trained on the circles dataset.



(a) Layer 1 (b) Layer 2 (c) Layer 3 (d) Layer 4 (e) Layer 5 (f) Layer 6 (g) Layer 7 (h) Layer 8 (i) Layer 9 (j) Layer 10

Fig. 17. Node-specific activation of input space for the network with $n = 10, H = 10$ and *tanh* as the activation function trained on the spirals dataset. Blue denotes activation of -1, and yellow denotes activation of 1. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

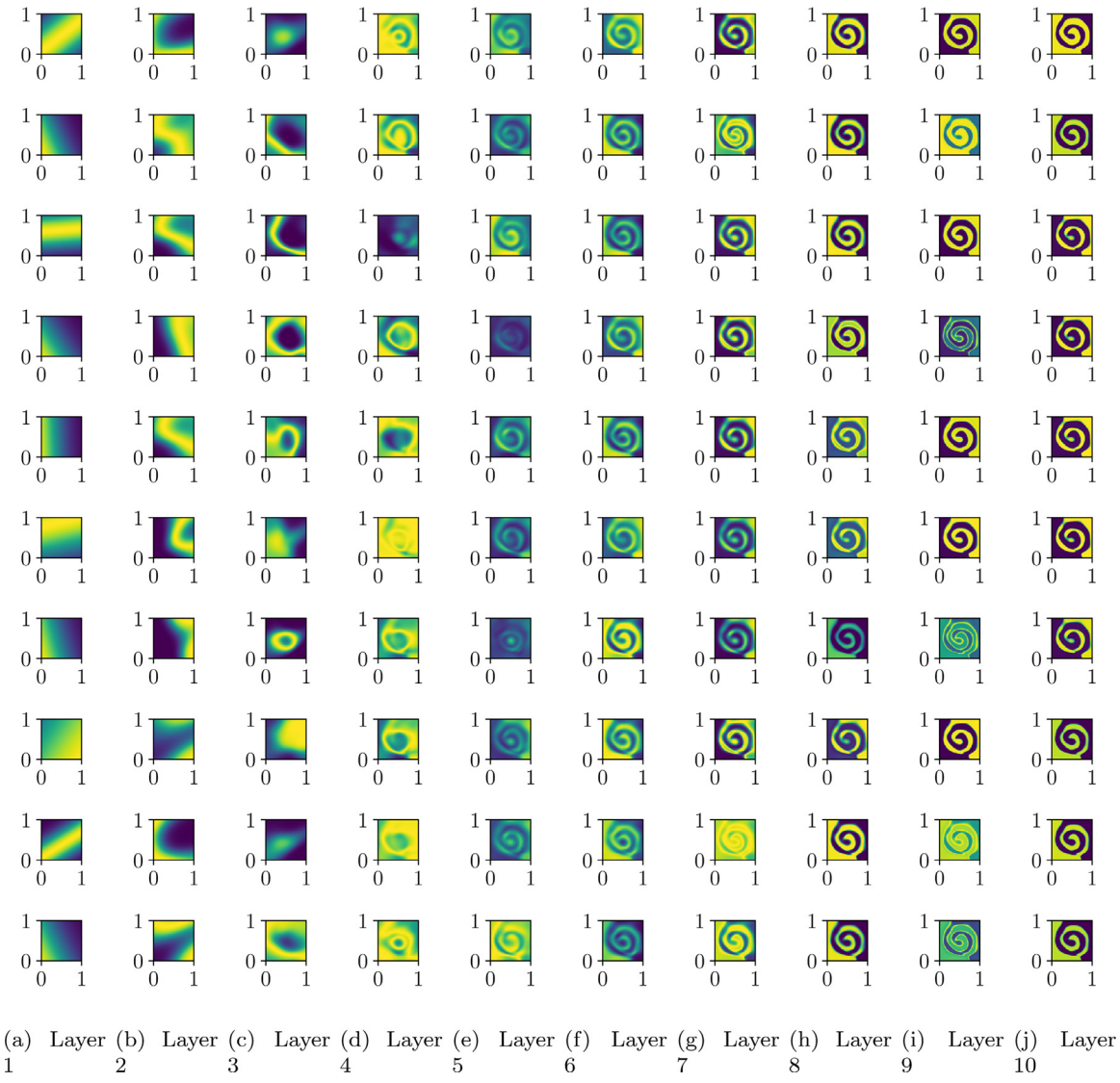


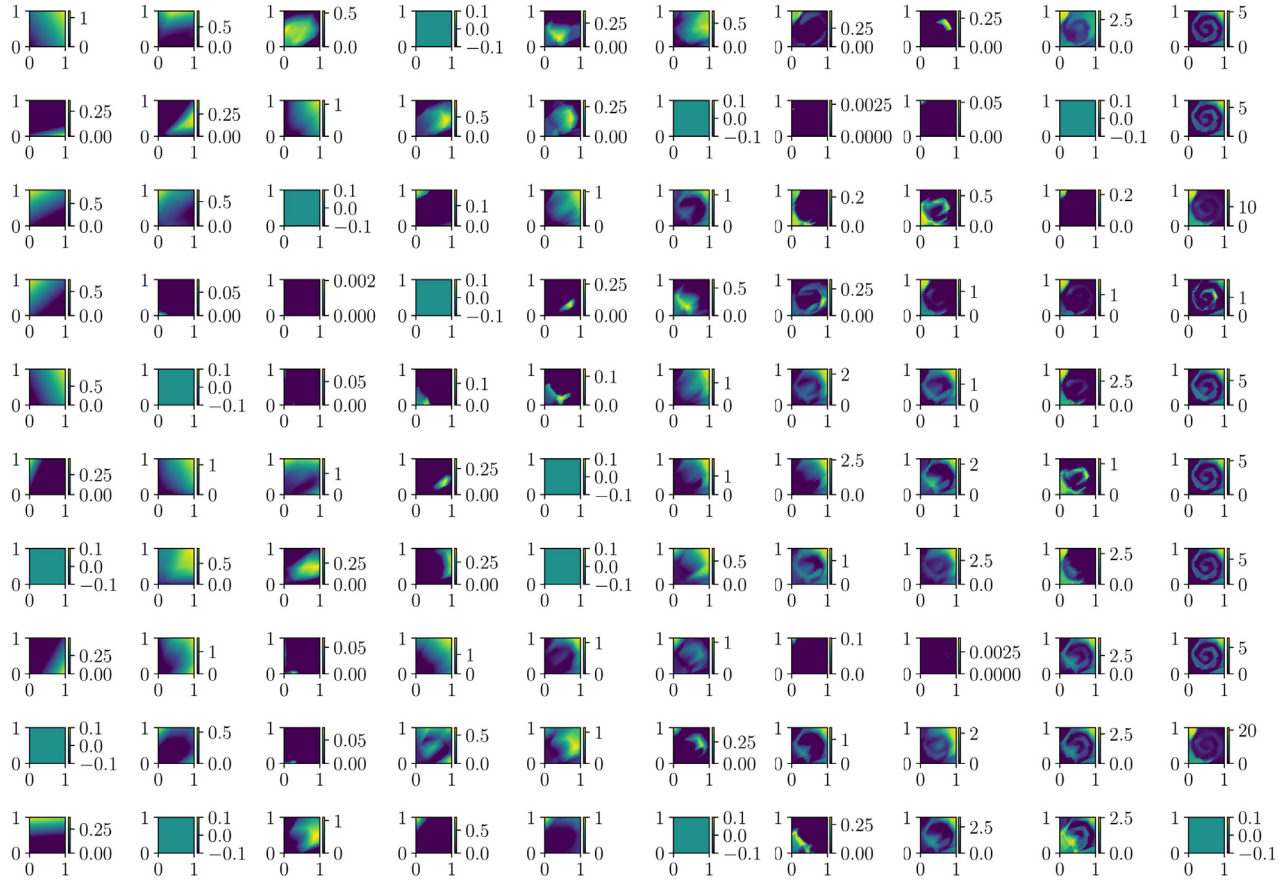
Fig. 18. Node-specific activation of input space for the network with $n = 10, H = 10$ and *Gaussian* activation function trained on the spirals dataset. Blue denotes activation of 0, and yellow denotes activation of 1. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

of the configurations of weights. It therefore follows that a network that is being trained with the dataset $\mathcal{D}$ can witness decreasing losses solely due to the increase in value of the weights in the final layer. However, this does not involve any modification of the decision boundaries, and hence any improvement in the accuracy of classification. In this context, the technique of weight normalization and regularization (Glorot and Bengio, 2010; Salimans and Kingma, 2016), that restricts the norm of the weight vector incoming on a single node to a value of one, and early stopping (Prechelt, 1998; Caruana et al., 2001) prevents such a scenario. It must be noted here that the impact of degeneracy of parameters on the input space of a layer is different for the hidden and final layers. Specifically, while the degenerate weights in the hidden layer result in different selective activations of the input space and hence different features extracted from the input, the degenerate weights in the final layer do not result in any change in the decision boundaries, and can provide a false sense of *better learning*.

In summary, we identified different design choices in building neural networks, including the number of nodes, number of layers and activation function that influence the performance of the network. We studied the contribution of each of these factors to the final output of the network with a systematic analysis of the role of each node, the number of layers and the number of nodes in the following. In the following, we discuss how these individual components are combined to achieve a particular classification task.

4. How does a neural network learn the mapping?: From parts to whole

In this section we merge the different components of the network and show how the network activates different segments from the input space, converting a nonlinear classification problem into a linear one. We consider three illustrative examples that highlight these features and an application of classification to fault diagnosis in a continuously stirred tank reactor. Fig. 10 shows the dataset for three different examples, i.e., two interlocked *moons*, two concentric *circles* and two *spirals*. In each case, we varied the number of nodes in hidden layers (n), number of hidden layer ($H = L - 1$) and the activation function ($g(\cdot)$) and trained networks with different combinations. The architectural details, classification loss and accuracy for all networks trained on the three examples are listed in Table 3. In each case, the network was trained with 80% of the dataset and tested on 20% of the dataset. Tuning of parameters



(a) Layer 1 (b) Layer 2 (c) Layer 3 (d) Layer 4 (e) Layer 5 (f) Layer 6 (g) Layer 7 (h) Layer 8 (i) Layer 9 (j) Layer 10

Fig. 19. Node-specific activation of input space for the network with $n = 10, H = 10$ and *ReLU* as the activation function trained on the spirals dataset.

Table 3
Consolidated results of different architectures after 1000 epochs of training.

Example	Activation	H	n	Loss	Accuracy
Moons	tanh	1	3	0.14	93%
		1	5	3×10^{-4}	100%
		2	3	3.6×10^{-7}	100%
		3	3	1.1×10^{-7}	100%
	ReLU	3	3	0.24	88%
		3	5	0.2	90%
	Gaussian	3	3	6×10^{-5}	100%
Circles	tanh	1	3	0.46	78%
		1	5	2.2×10^{-3}	100%
		2	3	4×10^{-7}	100%
		3	3	2.9×10^{-7}	100%
	ReLU	3	3	5×10^{-4}	100%
Spirals	tanh	1	3	0.62	59%
		1	10	0.65	61%
		1	50	0.65	61%
		1	100	0.61	61%
		3	10	0.61	60%
		3	100	0.65	61%
		10	10	10^{-5}	100%
	ReLU	10	10	3×10^{-4}	100%
	Gaussian	10	10	9×10^{-4}	100%
		3	100	6×10^{-5}	100%

was carried out with a momentum based gradient descent algorithm, i.e., ADAM implemented in `Keras`. It can be observed from Table 3 that the performance of the networks are highly sensitive to the architecture as well as the complexity of the problem. In order to further understand how the behaviour of the network as a whole is constituted from the individual elements discussed in the previous section, we study the transformation of feature space and selective activation performed by different networks.

We study the network with three hidden layers ($H = 3$) and three nodes in each hidden layer ($n = 3$) for the three illustrative examples. Figs. 11–13 show the feature space of the hidden and final layers of the networks trained on the moons dataset with *tanh, Gaussian* and *ReLU* as the activation function respectively. In each case, the interlocked moons can be observed to be untangled and progressively separated by successive layers of the network. It can be observed that for the same architecture, the three activation functions achieve the same objective by untangling the moons in different manners. For example, while the *tanh* and *Gaussian* activation functions can be observed to transform the input shape in a manner that clearly separates the two classes at the input to the third hidden layer, the network with *ReLU* as the activation function is unable the achieve adequate segregation between the classes. It can also be observed from Figs. 11 and 12 that the transformation achieved by these networks at the input to the third hidden layer allows identifying a separating hyperplane in the $\mathbf{z}^{[3]}-$ space for the networks. However, the third layer of the networks further transforms the space to increase the separation between the classes, and hence result in a lower value of loss (at the same classification accuracy). It can be inferred from these observations that increasing the depth of a network beyond a certain threshold, although yielding a decrease in the loss, does not

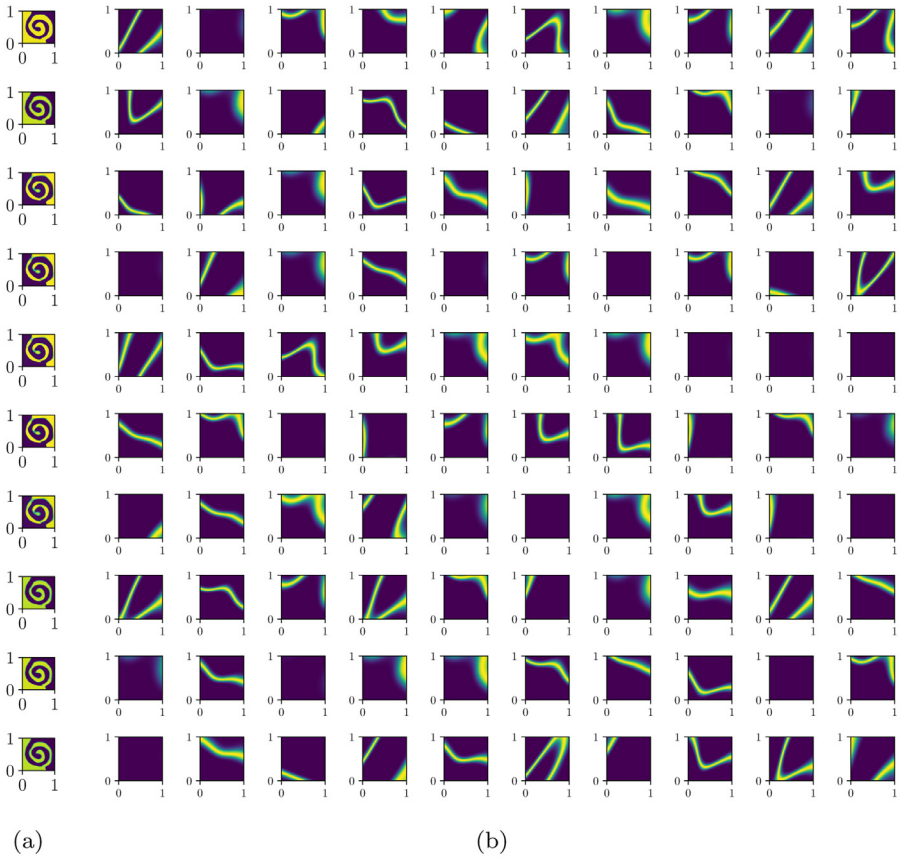


Fig. 20. A comparison of the final hidden layer activation for different architectures with Gaussian activation in example 3: Architecture (a) $n = 10$, $H = 10$ (b) $n = 100$, $H = 3$. Blue regions denote area in the input space where activation of the node becomes 0. Yellow denotes activation of 1. Notice how deeper networks yield complex activations while, shallow-wide networks have relatively simple activations. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

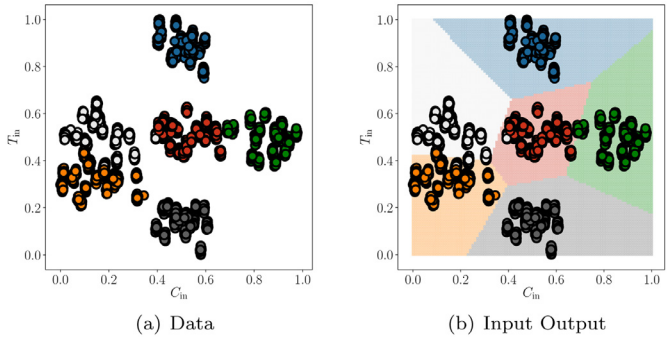


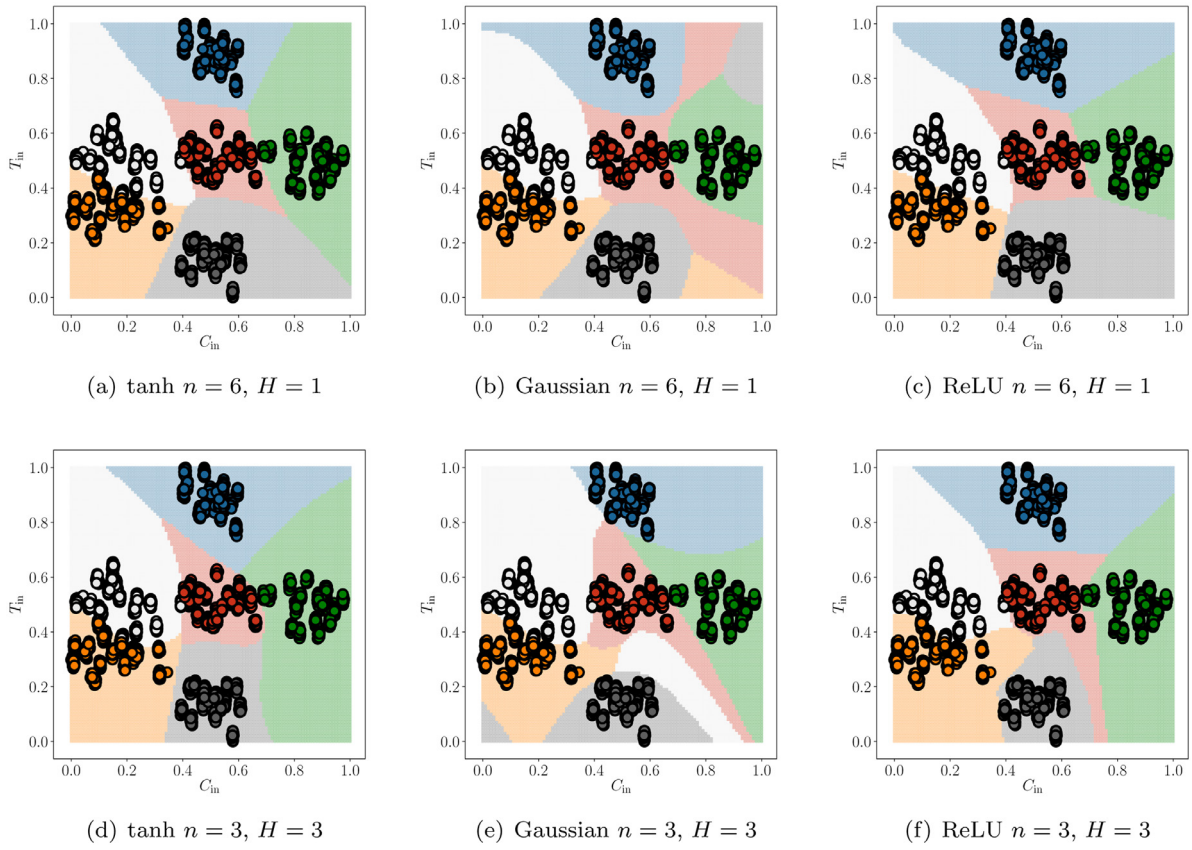
Fig. 21. (a) Fault space data (b) Fault space classification with a simple input output model.

necessarily imply an improvement in the internal representation of the network. Figs. 14–16 show the feature space of the layers of a network trained on the circle dataset with *tanh, Gaussian* and *ReLU* as the activation function respectively. It can be observed that the same network architecture now results in a different transformation of the input space that results in the separation of the two concentric circles. In addition, the two classes can be observed to be separable at the input to the third layer (for the networks with *tanh* and *Gaussian* as the activation function) while the additional layer serves to increase the distance between the two classes, thereby resulting in a lower loss.

It can be observed from Table 3 that networks trained on the spiral dataset require more resources than those trained on the moons and circles dataset for approximately same levels of performance. This is because of the highly intertwined nature of the dataset that results in a highly nonlinear separating boundary between the classes. Figs. 17–19 show the node-sensitive selective activation of the networks with $n = 10, H = 10$ and *tanh, Gaussian* and *ReLU* as the activation function respectively. Each plot in Figs. 17–19 represents the selective activation performed by one node in the network, with the columns representing different layers and rows representing different nodes in one layer. It can be observed from Figs. 17–19 that the nodes in initial layers of the network identify simple patterns on the input space, while the final layers operate on these simple patterns to generate complex shapes, most of which closely resemble the spiral distribution of the data in the input space.

Fig. 20 compares the selective activation performed by the final hidden layer of the networks with $n = 10, H = 10$ and $n = 100, H = 3$, which highlights the difference between the two representations. Specifically, final hidden layer of the deep network reveals complex geometrical patterns constructed on the input space, while the shallow and wide network identifies very simple patterns on the input space. This verifies the observation made in the earlier sections regarding the impact of depth and width of the network on the complexity of features extracted by the network.

The above discussion highlighted the transformation of input space performed by each layer in the network as well as the selective activation performed by the individual nodes in a layer. It was observed that all the nodes in the network learn (tune their parameters) towards a common objective of minimising the loss and in the process (i) transform the space and (ii) identify selec-



(a) tanh $n = 6$, $H = 1$ (b) Gaussian $n = 6$, $H = 1$ (c) ReLU $n = 6$, $H = 1$

(d) tanh $n = 3$, $H = 3$ (e) Gaussian $n = 3$, $H = 3$ (f) ReLU $n = 3$, $H = 3$

Fig. 22. Fault space classification for different activation functions and architectures.

tive activations to separate the two classes. We also highlighted the influence of width and depth on the internal representation of the network. We next discuss the application of classification to fault diagnosis in chemical plants and examine the feature spaces and activations identified by the neural networks.

One of the most common applications of classification in chemical engineering involves diagnosing the operation of the plant as being normal or abnormal. and has been extensively studied in the literature (Venkatasubramanian and Chan, 1989; Watanabe et al., 1989; Hoskins et al., 1991; Fan et al., 1993; Askarian et al., 2016; Jiang et al., 2016; Shokry et al., 2017; Gharahbagheri et al., 2017; Zhang and Zhao, 2017; Wu and Zhao, 2018). This involves employing the measurements obtained from sensors installed in the plant and identify the same to belong one of many classes with each class representing a state of operation. The measurements from sensors along with appropriate labels therefore constitute the dataset used for training neural network based classifier.

It is useful to understand how the neural network internally represents and classifies the different normal and abnormal regions - i.e., peer into the black-box to gain some insights. The earliest work in this regard in the chemical engineering literature was by Vaidyanathan and Venkatasubramanian (1992) who studied the fault space classification structure of a CSTR process. We continue that effort further here to gain more insights by using the more modern tools available now.

We consider a exothermic first order continuous stirred tank reactor, with temperature control, that operates according the following equations:

$$\text{Concentration} \quad \frac{dC}{dt} = \frac{Q}{V}(C_{in} - C) - kC\exp\left(-\frac{E_a}{RT}\right)$$

$$\text{Temperature} \quad \frac{dT}{dt} = \frac{Q}{V}(T_{in} - T) + \frac{-\Delta H_{rxn}}{\rho c_p}kC\exp\left(-\frac{E_a}{RT}\right) - \frac{UA}{\rho c_p V}(T - T_c)$$

$$\text{Coolant temperature} \quad \frac{dT_c}{dt} = \frac{Q_c}{V_c}(T_{c,in} - T_c) + \frac{UA}{\rho_c c_{p,c} V_c}(T - T_c)$$

and that has two measurements – concentration of feed C_{in} and temperature of the inlet T_{in} that can be used to perform fault diagnosis. The process is simulated using the model in Pilario (2019) with disturbances in the feed concentration as well as inlet temperature under normal operating conditions. The following faults are then introduced into the reactor (in addition to disturbances) by changing the appropriate parameters of the system:

1. Normal operation
2. Fault 1: High inlet concentration
3. Fault 2: Low inlet concentration
4. Fault 3: High inlet temperature
5. Fault 4: Low inlet temperature
6. Fault 5: Low inlet concentration with moderately low inlet temperature

We generated 1200 samples of data for each class and performed training with 70% of the data, which was first used to train an input-output network that has no hidden layers. Fig. 21 shows the normalised data of concentration and temperature belonging to different classes and the separating boundaries of the input-output-network. It can be observed from Fig. 21 that, as expected, the input-output network identifies linear boundaries that separate the different classes, and deliver an accuracy of 96%. Since we now

Table 4
Fault diagnosis example – Consolidated results after 1000 epochs of training on the test sample.

Activation	H	n	Loss	Accuracy
-	0	0	0.1	96%
tanh	3	3	0.01	99%
Gaussian	3	3	0.03	98.8%
ReLU	3	3	0.009	99.7%
tanh	1	6	0.03	98.8%
Gaussian	1	6	0.03	99%
ReLU	1	6	0.037	98.7%

know that deep networks tend to make complicated features, and shallow-wide networks tend to exhibit a more varied set of final representation, we want to observe a combined effect of changing the width and depth simultaneously. We hence, trained networks with $n = 3, H = 3$ and $n = 6, H = 1$ and *tanh, Gaussian* and *ReLU* as activation functions to study the combined impact of depth and width, for each activation function on the performance of the network. The performance of all the models are listed in Table 4 while the separating boundaries identified by the different networks are highlighted in Fig. 22 for the deep and shallow networks with different activation functions. It can be observed from Fig. 22 that the networks with *tanh* as the activation function tend to identify simple and reasonable separating boundaries, while the networks with *Gaussian* activation function identify fairly complex boundaries. The boundaries identified by the network with $n = 3, H = 3$ and *ReLU* as the activation function also exhibit a similar behavior. In addition, it can also be observed that the networks with *Gaussian* activation function result in disjoint segments of the input space being identified as belonging to the same class. It is noteworthy here that all these boundaries result in a classification accuracy of at least 98.7% and yet exhibit internal representations that are neither an accurate representation of the truth, nor robust to noise in the data. For example, one can easily identify regions in the input space which do not belong to the normal class (shaded in red) which will however be classified with exceedingly high probabilities as representative of healthy operation of the reactor. These regions and the corresponding data points constitute *adversarial examples* (Dalvi et al., 2004; Goodfellow et al., 2014; Kurakin et al., 2016; Gu and Rigazio, 2014) that pose a significant threat to the reliability of the network. This is in contrast with the input-output network and the networks with *tanh* as the activation function that do not exhibit such behaviours.

5. Conclusion

Deep neural networks have evolved into a versatile and powerful tool applicable for a wide range of problems. However, a clear understanding of their internal mechanism that allows for interpretations of internal representations and the parameters involved has not been developed completely yet. In this paper, we presented our findings from a systematic approach to understanding the operation of neural networks for classification tasks. We began with an individual node and identified a *characteristic equation* that dictates the nature of the node-specific selective activation of the input space. We then studied how several of these activations are processed by nodes in the successive layers to generate complex patterns whose characteristic equations represent *nonlinear* curved geometric objects. We showed how wider networks result in several simple characteristic equations and hence simple patterns identified on the input space, while deeper networks result in complex patterns in the input space. Although both types of networks are capable of achieving comparable levels of performance, each has its own advantages and disadvantages. Wider networks result in simple patterns which are fairly generic and can potentially be used for multiple tasks. However, wider networks result in a large number of parameters as compared to deeper networks with comparable performance. On the other hand, deeper networks result in complex shapes that cannot be used as generic templates of patterns.

We also studied the transformation of input space by each layer and examined the degeneracy of parameters in the final layer. Specifically, we identified that there exist an infinite number of configuration of weights of the classifying layer that will result in the same classification accuracy. We also identified that while the weights of hidden layers also exhibit degeneracy, implying that the different weights result in the same characteristic curve on the input space, they result in different activations of the space relative to the characteristic curve and hence different internal representations. All the above entities constitute the individual components of a neural network, which when working together towards minimising the loss function, result in relevant features being extracted from the input space. These are then used for classification of the input data.

We also identified the source of techniques such as transfer learning, weight normalisation and early stopping that are employed to improve the convergence of training algorithms. We also identified the origin of adversarial examples in neural networks, which result in incorrect interpretations, and in safety-critical applications they can lead to concerns. These behaviors and properties were illustrated with three standard classification examples - the intertwined moons, concentric circles and spirals. Their implications for a fault diagnosis task were demonstrated with a continuous stirred tank reactor. In all the cases, it was observed that building deeper networks, while possibly resulting in a decrease of the loss function, does not necessarily imply a better learning or representation of the data. We also observed that networks with more hidden layers are more likely to identify arbitrarily shaped patterns that need not be representative of the true structure of the data, leading to adversarial challenges.

This paper is an initial step towards a systematic approach to understanding the internal mechanism of deep neural networks, and to uncover key hidden properties that can be exploited to improve such networks. Future work in this direction would involve understanding how neural networks perform regression with internal representations. The goal is to elicit key insights for modeling engineering systems with neural networks in a manner that allows one to assign physical meaning to the internal representations and/or to be able to combine them with first-principles knowledge.

Notation

Vectors are bolded lower case. Matrices are bolded upper case. Elements not bolded are scalars.

If superscript (i)/$[l]$ are absent the quantity is assumed to be for the corresponding sample/layer respectively.

N	number of samples in the dataset
m	dimension of the input
K	dimension of the outputx
L	number of layers in the network (excluding input and output)
H	number of hidden layers in the network ($= L - 1$)
n_l	number of nodes in layer l of the neural network
$\mathbf{x}^{(i)}$	$\in \mathbb{R}^m$ ith input sample, $\mathbf{x}^{(i)} = [x_1^{(i)}, x_2^{(i)}, \ldots, x_m^{(i)}]$
$\mathbf{y}^{(i)}$	$\in \mathbb{R}^K$ ith output sample, $\mathbf{y}^{(i)} = [y_1^{(i)}, y_2^{(i)}, \ldots, y_K^{(i)}]$
$\hat{\mathbf{y}}^{(i)}$	$\in \mathbb{R}^K$, ith predicted output
$\mathbf{w}^{[l]}$	$\in \mathbb{R}^{n_{l-1}}$, weights connecting a node in layer $l-1$ to layer l, $\mathbf{w}^{[l]} = [w_1^{[l]}, w_2^{[l]}, \ldots, w_{n_{l-1}}^{[l]}]^\top$

$\mathbf{W}^{[l]}$ $\in \mathbb{R}^{n_l \times n_{l-1}}$ Weight matrix connecting layer $l-1$ to layer l, note that $\mathbf{W}^{[l]} = \begin{bmatrix} -\mathbf{w}_1^{[l]\top}- \\ -\mathbf{w}_2^{[l]\top}- \\ \vdots \\ -\mathbf{w}_{n_l}^{[l]\top}- \end{bmatrix}$

$\mathbf{b}^{[l]}$ $\in \mathbb{R}^{n_l}$ bias vector of the nodes in layer l, $\mathbf{b}^{[l]} = [b_1^{[l]}, b_2^{[l]}, \ldots, b_{n_l}^{[l]}]^\top$

$\mathbf{a}^{[l]}$ $\in \mathbb{R}^{n_l}$ activation vector of nodes in layer l, codomain dependent on the activation

$\mathbf{z}^{[l]}$ $\in \mathbb{R}^{n_l}$ weighted sum of the inputs to nodes in layer l

$g^{[l]}(\cdot)$ Activation function of layer l

Declaration of Competing Interest

None.

Supplementary material

Supplementary material associated with this article can be found, in the online version, at doi:10.1016/j.compchemeng.2019.106669.

References

Askarian, M., Escudero, G., Graells, M., Zarghami, R., Jalali-Farahani, F., Mostoufi, N., 2016. Fault diagnosis of chemical processes with incomplete observations: a comparative study. Comput. Chem. Eng. 84, 104–116.

Caruana, R., Lawrence, S., Giles, C.L., 2001. Overfitting in neural nets: backpropagation, conjugate gradient, and early stopping. In: Advances in Neural Information Processing Systems, pp. 402–408.

Dalvi, N., Domingos, P., Sanghai, S., Verma, D., et al., 2004. Adversarial classification. In: Proceedings of the tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM, pp. 99–108.

Fan, J., Nikolaou, M., White, R.E., 1993. An approach to fault diagnosis of chemical processes via neural networks. AIChE J. 39 (1), 82–88.

Gharahbagheri, H., Imtiaz, S., Khan, F., 2017. Root cause diagnosis of process fault using kpca and bayesian network. Ind. Eng. Chem. Res. 56 (8), 2054–2070.

Glorot, X., Bengio, Y., 2010. Understanding the difficulty of training deep feedforward neural networks. In: Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, pp. 249–256.

Goodfellow, I.J., Shlens, J., Szegedy, C., 2014. Explaining and harnessing adversarial examples. arXiv: 1412.6572.

Gopalakrishnan, K., Khaitan, S.K., Choudhary, A., Agrawal, A., 2017. Deep convolutional neural networks with transfer learning for computer vision-based data–driven pavement distress detection. Construct. Build. Mater. 157, 322–330.

Gu, S., Rigazio, L., 2014. Towards deep neural network architectures robust to adversarial examples. arXiv: 1412.5068.

Hoskins, J., Kaliyur, K., Himmelblau, D.M., 1991. Fault diagnosis in complex chemical plants using artificial neural networks. AIChE J. 37 (1), 137–141.

Hutson, M., 2018. Ai researchers allege that machine learning is alchemy. Science 360 (6388), 861.

Jiang, P., Hu, Z., Liu, J., Yu, S., Wu, F., 2016. Fault diagnosis based on chemical sensor data with an active deep neural network. Sensors 16 (10), 1695.

Jordan, M., 2018. Artificial intelligence the revolution hasn't happened yet. Medium. Apr 19.

Kurakin, A., Goodfellow, I., Bengio, S., 2016. Adversarial examples in the physical world. arXiv:1607.02533v1.

Olden, J.D., Jackson, D.A., 2002. Illuminating the "black box": a randomization approach for understanding variable contributions in artificial neural networks. Ecol. Modell. 154 (1–2), 135–150.

Pan, S.J., Yang, Q., 2009. A survey on transfer learning. IEEE Trans. Knowl. Data Eng. 22 (10), 1345–1359.

Pilario, K. E., 2019. Feedback-controlled cstr process for fault simulation. MATLAB Central File Exchange, Retrieved June 10, 2019, (https://www.mathworks.com/matlabcentral/fileexchange/66189-feedback-controlled-cstr-process-for-fault-simulation).

Prechelt, L., 1998. Early stopping-but when? In: Neural Networks: Tricks of the trade. Springer, pp. 55–69.

Salimans, T., Kingma, D.P., 2016. Weight normalization: a simple reparameterization to accelerate training of deep neural networks. In: Advances in Neural Information Processing Systems, pp. 901–909.

Setiono, R., Liu, H., 1995. Understanding neural networks via rule extraction. In: IJCAI, 1, pp. 480–485.

Shin, H.-C., Roth, H.R., Gao, M., Lu, L., Xu, Z., Nogues, I., Yao, J., Mollura, D., Summers, R.M., 2016. Deep convolutional neural networks for computer-aided detection: Cnn architectures, dataset characteristics and transfer learning. IEEE Trans. Med. Imaging 35 (5), 1285–1298.

Shokry, A., Ardakani, M.H., Escudero, G., Graells, M., Espuña, A., 2017. Dynamic kriging based fault detection and diagnosis approach for nonlinear noisy dynamic processes. Comput. Chem. Eng. 106, 758–776.

Shwartz-Ziv, R., Tishby, N., 2017. Opening the black box of deep neural networks via information. arXiv:1703.00810v1.

Talo, M., Baloglu, U.B., Yıldırım, Ö., Acharya, U.R., 2019. Application of deep transfer learning for automated brain abnormality classification using mr images. Cognit. Syst. Res. 54, 176–188.

Taylor, M.E., Stone, P., 2009. Transfer learning for reinforcement learning domains: a survey. J. Mach. Learn. Res. 10 (Jul), 1633–1685.

Thiese, M.S., Arnold, Z.C., Walker, S.D., 2015. The misuse and abuse of statistics in biomedical research. Biochem. Med. 25 (1), 5–11.

Tishby, N., Pereira, F. C., Bialek, W., 2000. The information bottleneck method. arXiv:physics/0004057.

Tishby, N., Zaslavsky, N., 2015. Deep learning and the information bottleneck principle. In: 2015 IEEE Information Theory Workshop (ITW). IEEE, pp. 1–5.

Vaidyanathan, R., Venkatasubramanian, V., 1992. On the nature of fault space classification structure developed by neural networks. Eng. Appl. Artif.Intell. 5 (4), 289–297.

Venkatasubramanian, V., 2019. The promise of artificial intelligence in chemical engineering: is it here, finally? AIChE J. 65 (2), 466–478.

Venkatasubramanian, V., Chan, K., 1989. A neural network methodology for process fault diagnosis. AIChE J. 35 (12), 1993–2002.

Watanabe, K., Matsuura, I., Abe, M., Kubota, M., Himmelblau, D., 1989. Incipient fault diagnosis of chemical processes via artificial neural networks. AIChE J. 35 (11), 1803–1812.

Weiss, K., Khoshgoftaar, T.M., Wang, D., 2016. A survey of transfer learning. J. Big data 3 (1), 9.

Wu, H., Zhao, J., 2018. Deep convolutional neural network model based chemical process fault diagnosis. Comput. Chem. Eng. 115, 185–197.

Yosinski, J., Clune, J., Nguyen, A., Fuchs, T., Lipson, H., 2015. Understanding neural networks through deep visualization. arXiv:1506.06579v1.

Zhang, Z., Zhao, J., 2017. A deep belief network based fault diagnosis model for complex chemical processes. Comput. Chem. Eng. 107, 395–407.