Chapter 4

Deep neural networks

The last chapter described shallow neural networks, which have a single hidden layer. This chapter introduces deep neural networks, which have more than one hidden layer. With ReLU activation functions, both shallow and deep networks describe piecewise linear mappings from input to output.

As the number of hidden units increases, shallow neural networks improve their descriptive power. Indeed, with enough hidden units, shallow networks can describe arbitrarily complex functions in high dimensions. However, it turns out that for some functions, the required number of hidden units is impractically large. Deep networks can produce many more linear regions than shallow networks for a given number of parameters. Hence, from a practical standpoint, they can be used to describe a broader family of functions.

4.1 Composing neural networks

To gain insight into the behavior of deep neural networks, we first consider composing two shallow networks so the output of the first becomes the input of the second. Consider two shallow networks with three hidden units each (figure 4.1a). The first network takes an input x and returns output y and is defined by:

$$\begin{aligned} h_1 &= \mathrm{a}[\theta_{10} + \theta_{11}x] \\ h_2 &= \mathrm{a}[\theta_{20} + \theta_{21}x] \\ h_3 &= \mathrm{a}[\theta_{30} + \theta_{31}x], \end{aligned} \tag{4.1}$$

and

$$y = \phi_0 + \phi_1 h_1 + \phi_2 h_2 + \phi_3 h_3. \tag{4.2}$$

The second network takes y as input and returns y' and is defined by:

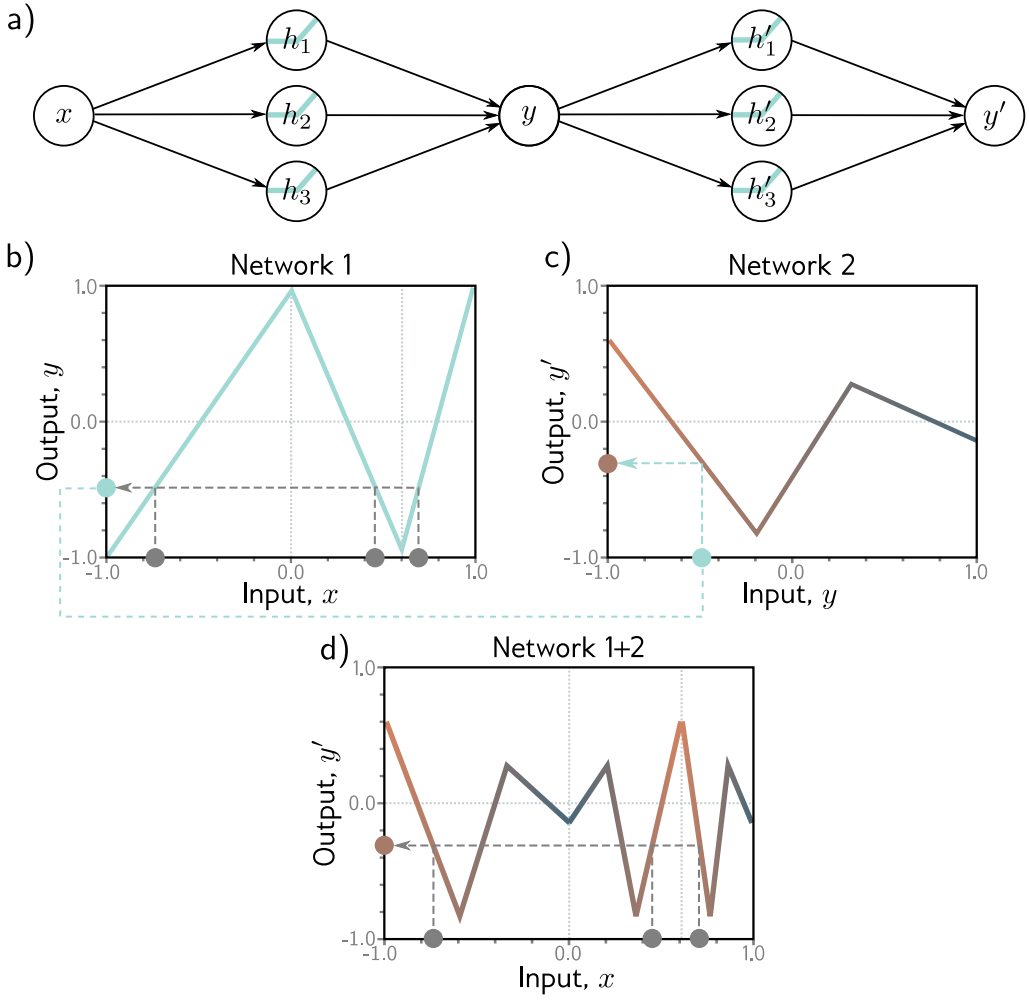

Figure 4.1 Composing two single-layer networks with three hidden units each. a) The output y of the first network constitutes the input to the second network. b) The first network maps inputs $x \in [-1, 1]$ to outputs $y \in [-1, 1]$ using a function comprising three linear regions that are chosen so that they alternate the sign of their slope (fourth linear region is outside range of graph). Multiple inputs x (gray circles) now map to the same output y (cyan circle). c) The second network defines a function comprising three linear regions that takes y and returns y' (i.e., the cyan circle is mapped to the brown circle). d) The combined effect of these two functions when composed is that (i) three different inputs x are mapped to any given value of y by the first network and (ii) are processed in the same way by the second network; the result is that the function defined by the second network in panel (c) is duplicated three times, variously flipped and rescaled according to the slope of the regions of panel (b).

$$\begin{aligned} h'_1 &= \mathrm{a}[\theta'_{10} + \theta'_{11} y] \\ h'_2 &= \mathrm{a}[\theta'_{20} + \theta'_{21} y] \\ h'_3 &= \mathrm{a}[\theta'_{30} + \theta'_{31} y], \end{aligned} \tag{4.3}$$

and

$$y' = \phi'_0 + \phi'_1 h'_1 + \phi'_2 h'_2 + \phi'_3 h'_3. \tag{4.4}$$

With ReLU activations, this model also describes a family of piecewise linear functions. However, the number of linear regions is potentially greater than for a shallow network with six hidden units. To see this, consider choosing the first network to produce three alternating regions of positive and negative slope (figure 4.1b). This means that three different ranges of x are mapped to the same output range $y \in [-1, 1]$, and the subsequent mapping from this range of y to y' is applied three times. The overall effect is that the function defined by the second network is duplicated three times to create nine linear regions. The same principle applies in higher dimensions (figure 4.2).

Problem 4.1

Notebook 4.1 Composing networks

A different way to think about composing networks is that the first network "folds" the input space x back onto itself so that multiple inputs generate the same output. Then the second network applies a function, which is replicated at all points that were folded on top of one another (figure 4.3).

4.2 From composing networks to deep networks

The previous section showed that we could create complex functions by passing the output of one shallow neural network into a second network. We now show that this is a special case of a deep network with two hidden layers.

The output of the first network ($y = \phi_0 + \phi_1 h_1 + \phi_2 h_2 + \phi_3 h_3$) is a linear combination of the activations at the hidden units. The first operations of the second network (equation 4.3 in which we calculate $\theta'_{10} + \theta'_{11} y$, $\theta'_{20} + \theta'_{21} y$, and $\theta'_{30} + \theta'_{31} y$) are linear in the output of the first network. Applying one linear function to another yields another linear function. Substituting the expression for y into equation 4.3 gives:

$$\begin{aligned} h'_1 &= \mathrm{a}[\theta'_{10} + \theta'_{11} y] &&= \mathrm{a}[\theta'_{10} + \theta'_{11}\phi_0 + \theta'_{11}\phi_1 h_1 + \theta'_{11}\phi_2 h_2 + \theta'_{11}\phi_3 h_3] \\ h'_2 &= \mathrm{a}[\theta'_{20} + \theta'_{21} y] &&= \mathrm{a}[\theta'_{20} + \theta'_{21}\phi_0 + \theta'_{21}\phi_1 h_1 + \theta'_{21}\phi_2 h_2 + \theta'_{21}\phi_3 h_3] \\ h'_3 &= \mathrm{a}[\theta'_{30} + \theta'_{31} y] &&= \mathrm{a}[\theta'_{30} + \theta'_{31}\phi_0 + \theta'_{31}\phi_1 h_1 + \theta'_{31}\phi_2 h_2 + \theta'_{31}\phi_3 h_3], \end{aligned} \tag{4.5}$$

which we can rewrite as:

$$\begin{aligned} h'_1 &= \mathrm{a}[\psi_{10} + \psi_{11} h_1 + \psi_{12} h_2 + \psi_{13} h_3] \\ h'_2 &= \mathrm{a}[\psi_{20} + \psi_{21} h_1 + \psi_{22} h_2 + \psi_{23} h_3] \\ h'_3 &= \mathrm{a}[\psi_{30} + \psi_{31} h_1 + \psi_{32} h_2 + \psi_{33} h_3], \end{aligned} \tag{4.6}$$

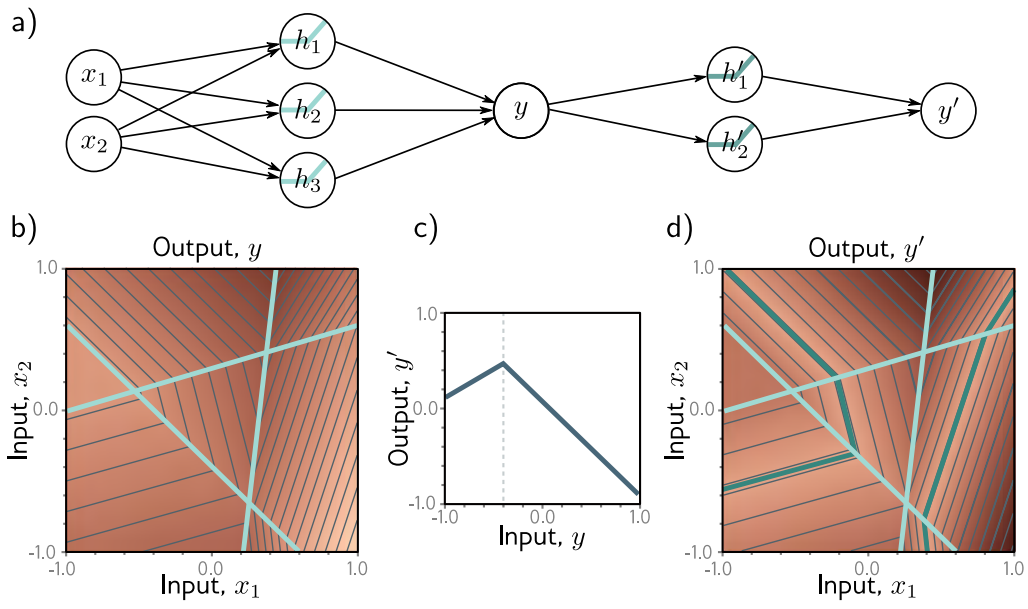

Figure 4.2 Composing neural networks with a 2D input. a) The first network (from figure 3.8) has three hidden units and takes two inputs x_1 and x_2 and returns a scalar output y. This is passed into a second network with two hidden units to produce y'. b) The first network produces a function consisting of seven linear regions, one of which is flat. c) The second network defines a function comprising two linear regions in $y \in [-1, 1]$. d) When these networks are composed, each of the six non-flat regions from the first network is divided into two new regions by the second network to create a total of 13 linear regions.

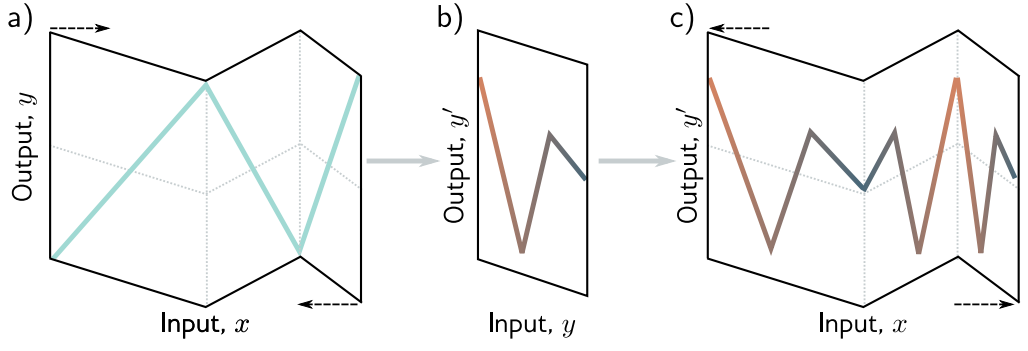

Figure 4.3 Deep networks as folding input space. a) One way to think about the first network from figure 4.1 is that it "folds" the input space back on top of itself. b) The second network applies its function to the folded space. c) The final output is revealed by "unfolding" again.

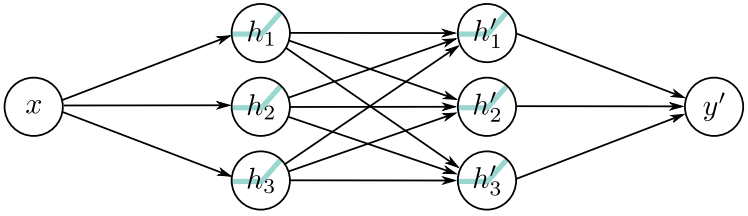

Figure 4.4 Neural network with one input, one output, and two hidden layers, each containing three hidden units.

where $\psi_{10} = \theta'_{10} + \theta'_{11}\phi_0, \psi_{11} = \theta'_{11}\phi_1, \psi_{12} = \theta'_{11}\phi_2$ and so on. The result is a network with two hidden layers (figure 4.4).

It follows that a network with two layers can represent the family of functions created by passing the output of one single-layer network into another. In fact, it represents a broader family because in equation 4.6, the nine slope parameters $\psi_{11}, \psi_{21}, \ldots, \psi_{33}$ can take arbitrary values, whereas, in equation 4.5, these parameters are constrained to be the outer product $[\theta'_{11}, \theta'_{21}, \theta'_{31}]^T[\phi_1, \phi_2, \phi_3]$.

4.3 Deep neural networks

In the previous section, we showed that composing two shallow networks yields a special case of a deep network with two hidden layers. Now we consider the general case of a deep network with two hidden layers, each containing three hidden units (figure 4.4). The first layer is defined by:

$$\begin{aligned} h_1 &= \mathrm{a}[\theta_{10} + \theta_{11}x] \\ h_2 &= \mathrm{a}[\theta_{20} + \theta_{21}x] \\ h_3 &= \mathrm{a}[\theta_{30} + \theta_{31}x], \end{aligned} \tag{4.7}$$

the second layer by:

$$\begin{aligned} h'_1 &= \mathrm{a}[\psi_{10} + \psi_{11}h_1 + \psi_{12}h_2 + \psi_{13}h_3] \\ h'_2 &= \mathrm{a}[\psi_{20} + \psi_{21}h_1 + \psi_{22}h_2 + \psi_{23}h_3] \\ h'_3 &= \mathrm{a}[\psi_{30} + \psi_{31}h_1 + \psi_{32}h_2 + \psi_{33}h_3], \end{aligned} \tag{4.8}$$

and the output by:

$$y' = \phi'_0 + \phi'_1 h'_1 + \phi'_2 h'_2 + \phi'_3 h'_3. \tag{4.9}$$

Notebook 4.2 Clipping functions

Considering these equations leads to another way to think about how the network constructs an increasingly complicated function (figure 4.5):

1. The three hidden units h_1, h_2, and h_3 in the first layer are computed as usual by forming linear functions of the input and passing these through ReLU activation functions (equation 4.7).
2. The pre-activations at the second layer are computed by taking three new linear functions of these hidden units (arguments of the activation functions in equation 4.8). At this point, we effectively have a shallow network with three outputs; we have computed three piecewise linear functions with the "joints" between linear regions in the same places (see figure 3.6).
3. At the second hidden layer, another ReLU function $\mathrm{a}[\bullet]$ is applied to each function (equation 4.8), which clips them and adds new "joints" to each.
4. The final output is a linear combination of these hidden units (equation 4.9).

In conclusion, we can either think of each layer as "folding" the input space or as creating new functions, which are clipped (creating new regions) and then recombined. The former view emphasizes the dependencies in the output function but not how clipping creates new joints, and the latter has the opposite emphasis. Ultimately, both descriptions provide only partial insight into how deep neural networks operate. Regardless, it's important not to lose sight of the fact that this is still merely an equation relating input x to output y'. Indeed, we can combine equations 4.7–4.9 to get one expression:

$$\begin{aligned} y' = \; & \phi_0' + \phi_1' \mathrm{a}\left[\psi_{10} + \psi_{11}\mathrm{a}[\theta_{10} + \theta_{11}x] + \psi_{12}\mathrm{a}[\theta_{20} + \theta_{21}x] + \psi_{13}\mathrm{a}[\theta_{30} + \theta_{31}x]\right] \\ & + \phi_2' \mathrm{a}[\psi_{20} + \psi_{21}\mathrm{a}[\theta_{10} + \theta_{11}x] + \psi_{22}\mathrm{a}[\theta_{20} + \theta_{21}x] + \psi_{23}\mathrm{a}[\theta_{30} + \theta_{31}x]] \\ & + \phi_3' \mathrm{a}[\psi_{30} + \psi_{31}\mathrm{a}[\theta_{10} + \theta_{11}x] + \psi_{32}\mathrm{a}[\theta_{20} + \theta_{21}x] + \psi_{33}\mathrm{a}[\theta_{30} + \theta_{31}x]], \end{aligned} \tag{4.10}$$

although this is admittedly rather difficult to understand.

4.3.1 Hyperparameters

We can extend the deep network construction to more than two hidden layers; modern networks might have more than a hundred layers with thousands of hidden units at each layer. The number of hidden units in each layer is referred to as the *width* of the network, and the number of hidden layers as the *depth*. The total number of hidden units is a measure of the network's *capacity*.

Problem 4.2

We denote the number of layers as K and the number of hidden units in each layer as $D_1, D_2, \ldots, D_K$. These are examples of *hyperparameters*. They are quantities chosen before we learn the model parameters (i.e., the slope and intercept terms). For fixed hyperparameters (e.g., $K = 2$ layers with $D_k = 3$ hidden units in each), the model describes a family of functions, and the parameters determine the particular function. Hence, when we also consider the hyperparameters, we can think of neural networks as representing a family of families of functions relating input to output.

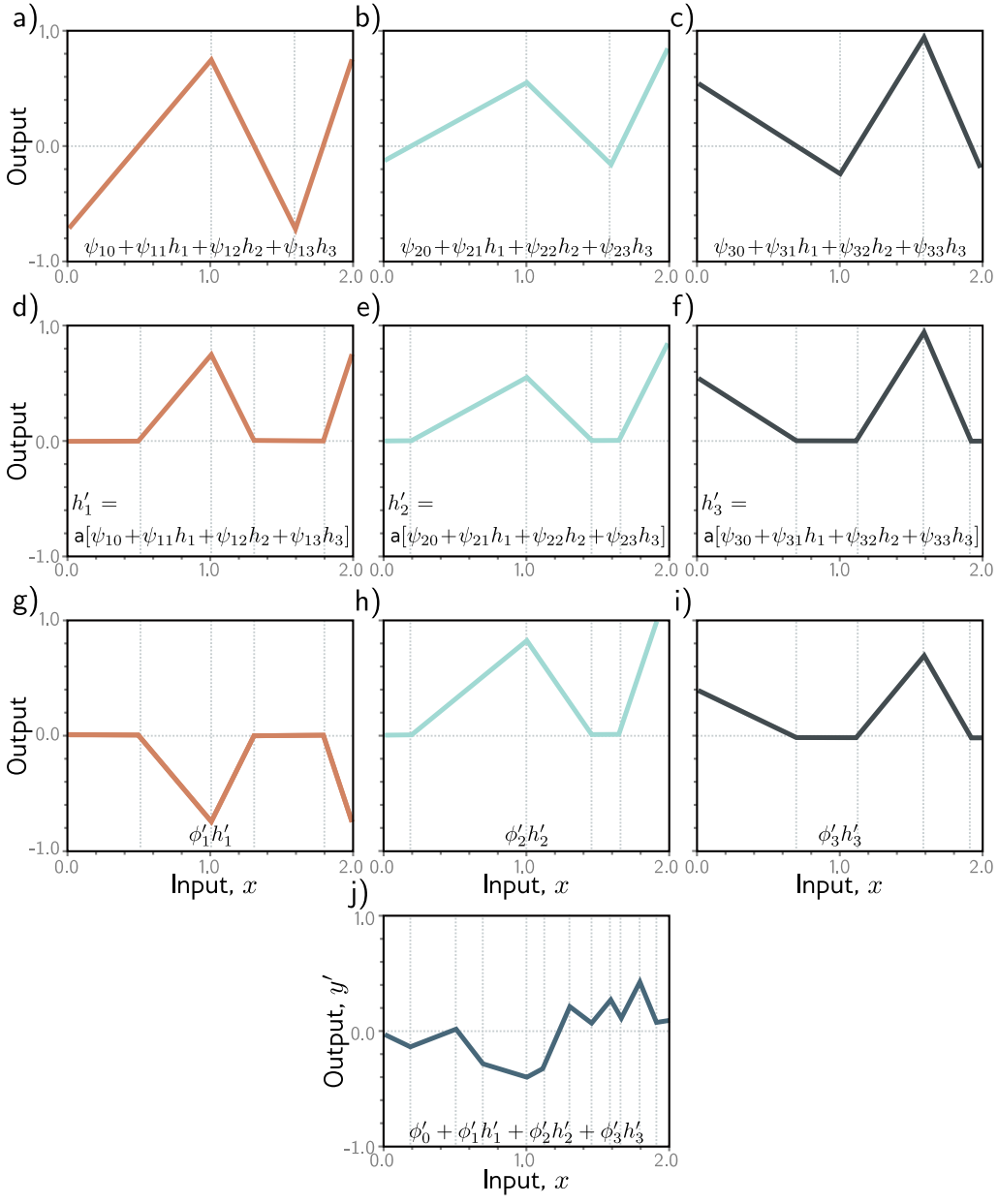

Figure 4.5 Computation for the deep network in figure 4.4. a–c) The inputs to the second hidden layer (i.e., the pre-activations) are three piecewise linear functions where the "joints" between the linear regions are at the same places (see figure 3.6). d–f) Each piecewise linear function is clipped to zero by the ReLU activation function. g–i) These clipped functions are then weighted with parameters ϕ'_1, ϕ'_2, and ϕ'_3, respectively. j) Finally, the clipped and weighted functions are summed and an offset ϕ'_0 that controls the overall height is added.

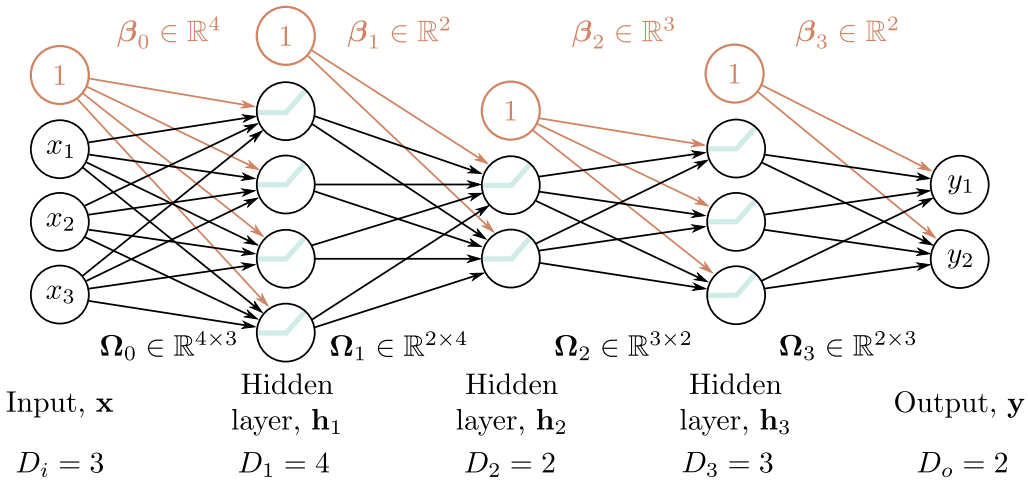

Figure 4.6 Matrix notation for network with $D_i = 3$-dimensional input $\mathbf{x}$, $D_o = 2$-dimensional output $\mathbf{y}$, and $K = 3$ hidden layers $\mathbf{h}_1, \mathbf{h}_2$, and $\mathbf{h}_3$ of dimensions $D_1 = 4$, $D_2 = 2$, and $D_3 = 3$ respectively. The weights are stored in matrices $\boldsymbol{\Omega}_k$ that multiply the activations from the preceding layer to create the pre-activations at the subsequent layer. For example, the weight matrix $\boldsymbol{\Omega}_1$ that computes the pre-activations at $\mathbf{h}_2$ from the activations at $\mathbf{h}_1$ has dimension 2×4. It is applied to the four hidden units in layer one and creates the inputs to the two hidden units at layer two. The biases are stored in vectors $\boldsymbol{\beta}_k$ and have the dimension of the layer into which they feed. For example, the bias vector $\boldsymbol{\beta}_2$ is length three because layer $\mathbf{h}_3$ contains three hidden units.

4.4 Matrix notation

Appendix B.3 Matrices

We have seen that a deep neural network consists of linear transformations alternating with activation functions. We could equivalently describe equations 4.7–4.9 in matrix notation as:

$$\begin{bmatrix} h_1 \\ h_2 \\ h_3 \end{bmatrix} = \mathbf{a}\left[\begin{bmatrix} \theta_{10} \\ \theta_{20} \\ \theta_{30} \end{bmatrix} + \begin{bmatrix} \theta_{11} \\ \theta_{21} \\ \theta_{31} \end{bmatrix} x\right], \tag{4.11}$$

$$\begin{bmatrix} h'_1 \\ h'_2 \\ h'_3 \end{bmatrix} = \mathbf{a}\left[\begin{bmatrix} \psi_{10} \\ \psi_{20} \\ \psi_{30} \end{bmatrix} + \begin{bmatrix} \psi_{11} & \psi_{12} & \psi_{13} \\ \psi_{21} & \psi_{22} & \psi_{23} \\ \psi_{31} & \psi_{32} & \psi_{33} \end{bmatrix} \begin{bmatrix} h_1 \\ h_2 \\ h_3 \end{bmatrix}\right], \tag{4.12}$$

and

$$y' = \phi'_0 + \begin{bmatrix} \phi'_1 & \phi'_2 & \phi'_3 \end{bmatrix} \begin{bmatrix} h'_1 \\ h'_2 \\ h'_3 \end{bmatrix}, \tag{4.13}$$

or even more compactly in matrix notation as:

$$\begin{aligned}\mathbf{h} &= \mathbf{a}\left[\boldsymbol{\theta}_0 + \boldsymbol{\theta} x\right] \\ \mathbf{h}' &= \mathbf{a}\left[\boldsymbol{\psi}_0 + \boldsymbol{\Psi}\mathbf{h}\right] \\ y' &= \phi_0' + \boldsymbol{\phi}'\mathbf{h}', \end{aligned} \tag{4.14}$$

where, in each case, the function $\mathbf{a}[\bullet]$ applies the activation function separately to every element of its vector input.

4.4.1 General formulation

This notation becomes cumbersome for networks with many layers. Hence, from now on, we will describe the vector of hidden units at layer k as $\mathbf{h}_k$, the vector of biases (intercepts) that contribute to hidden layer $k+1$ as $\boldsymbol{\beta}_k$, and the weights (slopes) that are applied to the k^{th} layer and contribute to the $(k+1)^{th}$ layer as $\boldsymbol{\Omega}_k$. A general deep network $\mathbf{y} = \mathrm{f}[\mathbf{x}, \boldsymbol{\phi}]$ with K layers can now be written as:

$$\begin{aligned}\mathbf{h}_1 &= \mathbf{a}[\boldsymbol{\beta}_0 + \boldsymbol{\Omega}_0\mathbf{x}] \\ \mathbf{h}_2 &= \mathbf{a}[\boldsymbol{\beta}_1 + \boldsymbol{\Omega}_1\mathbf{h}_1] \\ \mathbf{h}_3 &= \mathbf{a}[\boldsymbol{\beta}_2 + \boldsymbol{\Omega}_2\mathbf{h}_2] \\ &\vdots \\ \mathbf{h}_K &= \mathbf{a}[\boldsymbol{\beta}_{K-1} + \boldsymbol{\Omega}_{K-1}\mathbf{h}_{K-1}] \\ \mathbf{y} &= \boldsymbol{\beta}_K + \boldsymbol{\Omega}_K\mathbf{h}_K. \end{aligned} \tag{4.15}$$

The parameters $\boldsymbol{\phi}$ of this model comprise all of these weight matrices and bias vectors $\boldsymbol{\phi} = \{\boldsymbol{\beta}_k, \boldsymbol{\Omega}_k\}_{k=0}^{K}$.

If the k^{th} layer has D_k hidden units, then the bias vector $\boldsymbol{\beta}_{k-1}$ will be of size D_k. The last bias vector $\boldsymbol{\beta}_K$ has the size D_o of the output. The first weight matrix $\boldsymbol{\Omega}_0$ has size $D_1 \times D_i$ where D_i is the size of the input. The last weight matrix $\boldsymbol{\Omega}_K$ is $D_o \times D_K$, and the remaining matrices $\boldsymbol{\Omega}_k$ are $D_{k+1} \times D_k$ (figure 4.6).

Notebook 4.3
Deep networks

Problems 4.3–4.6

We can equivalently write the network as a single function:

$$\mathbf{y} = \boldsymbol{\beta}_K + \boldsymbol{\Omega}_K\mathbf{a}\left[\boldsymbol{\beta}_{K-1} + \boldsymbol{\Omega}_{K-1}\mathbf{a}\left[\ldots\boldsymbol{\beta}_2 + \boldsymbol{\Omega}_2\mathbf{a}\left[\boldsymbol{\beta}_1 + \boldsymbol{\Omega}_1\mathbf{a}\left[\boldsymbol{\beta}_0 + \boldsymbol{\Omega}_0\mathbf{x}\right]\right]\ldots\right]\right]. \tag{4.16}$$

4.5 Shallow vs. deep neural networks

Chapter 3 discussed shallow networks (with a single hidden layer), and here we have described deep networks (with multiple hidden layers). We now compare these models.

4.5.1 Ability to approximate different functions

In section 3.2, we argued that shallow neural networks with enough capacity (hidden units) could model any continuous function arbitrarily closely. In this chapter, we saw that a deep network with two hidden layers could represent the composition of two shallow networks. If the second of these networks computes the identity function, then this deep network replicates a single shallow network. Hence, it can also approximate any continuous function arbitrarily closely given sufficient capacity.

Problem 4.7

4.5.2 Number of linear regions per parameter

A shallow network with one input, one output, and $D > 2$ hidden units can create up to $D + 1$ linear regions and is defined by $3D + 1$ parameters. A deep network with one input, one output, and K layers of $D > 2$ hidden units can create a function with up to $(D + 1)^K$ linear regions using $3D + 1 + (K - 1)D(D + 1)$ parameters.

Problems 4.8–4.11

Figure 4.7a shows how the maximum number of linear regions increases as a function of the number of parameters for networks mapping scalar input x to scalar output y. Deep neural networks create much more complex functions for a fixed parameter budget. This effect is magnified as the number of input dimensions D_i increases (figure 4.7b), although computing the maximum number of regions is less straightforward.

This seems attractive, but the flexibility of the functions is still limited by the number of parameters. Deep networks can create extremely large numbers of linear regions, but these contain complex dependencies and symmetries. We saw some of these when we considered deep networks as "folding" the input space (figure 4.3). So, it's not clear that the greater number of regions is an advantage unless (i) there are similar symmetries in the real-world functions that we wish to approximate or (ii) we have reason to believe that the mapping from input to output really does involve a composition of simpler functions.

4.5.3 Depth efficiency

Both deep and shallow networks can model arbitrary functions, but some functions can be approximated much more efficiently with deep networks. Functions have been identified that require a shallow network with exponentially more hidden units to achieve an equivalent approximation to that of a deep network. This phenomenon is referred to as the *depth efficiency* of neural networks. This property is also attractive, but it's not clear that the real-world functions that we want to approximate fall into this category.

4.5.4 Large, structured inputs

We have discussed fully connected networks where every element of each layer contributes to every element of the subsequent one. However, these are not practical for large,

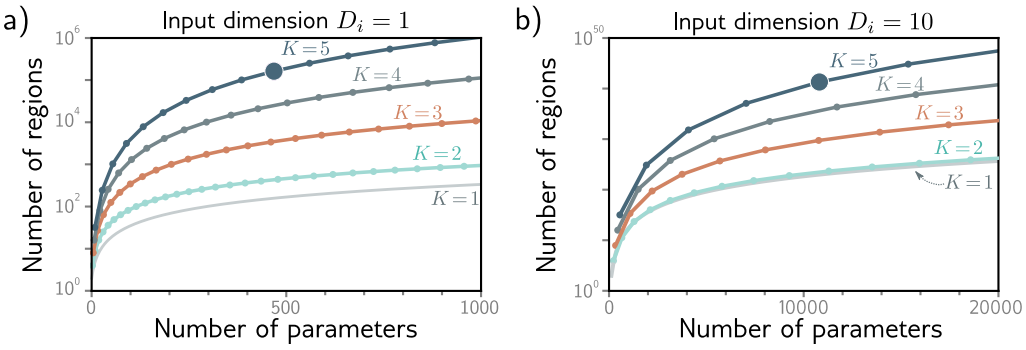

Figure 4.7 The maximum number of linear regions for neural networks increases rapidly with the network depth. a) Network with $D_i = 1$ input. Each curve represents a fixed number of hidden layers K, as we vary the number of hidden units D per layer. For a fixed parameter budget (horizontal position), deeper networks produce more linear regions than shallower ones. A network with $K = 5$ layers and $D = 10$ hidden units per layer has 471 parameters (highlighted point) and can produce 161,051 regions. b) Network with $D_i = 10$ inputs. Each subsequent point along a curve represents ten hidden units. Here, a model with $K = 5$ layers and $D = 50$ hidden units per layer has 10,801 parameters (highlighted point) and can create more than 10^{40} linear regions.

structured inputs like images, where the input might comprise $\sim 10^6$ pixels. The number of parameters would be prohibitive, and moreover, we want different parts of the image to be processed similarly; there is no point in independently learning to recognize the same object at every possible position in the image.

The solution is to process local image regions in parallel and then gradually integrate information from increasingly large regions. This kind of local-to-global processing is difficult to specify without using multiple layers (see chapter 10).

4.5.5 Training and generalization

A further possible advantage of deep networks over shallow networks is their ease of fitting; it is usually easier to train moderately deep networks than to train shallow ones (see figure 20.2). It may be that over-parameterized deep models (i.e., those with more parameters than training examples) have a large family of roughly equivalent solutions that are easy to find. However, as we add more hidden layers, training becomes more difficult again. Many methods have been developed to mitigate this problem (see chapter 11).

Deep neural networks also seem to generalize to new data better than shallow ones. In practice, the best results for most tasks have been achieved using networks with tens or hundreds of layers. Neither of these phenomena are well understood, and we return to them in chapter 20.

4.6 Summary

In this chapter, we first considered what happens when we compose two shallow networks. We argued that the first network "folds" the input space, and the second network then applies a piecewise linear function. The effects of the second network are duplicated where the input space is folded onto itself.

We then showed that this composition of shallow networks is a special case of a deep network with two layers. We interpreted the ReLU functions in each layer as clipping the input functions in multiple places and creating more "joints" in the output function. We introduced the idea of hyperparameters, which for the networks we've seen so far, comprise the number of hidden layers and the number of hidden units in each.

Finally, we compared shallow and deep networks. We saw that (i) both networks can approximate any function given enough capacity, (ii) deep networks produce many more linear regions per parameter, (iii) some functions can be approximated much more efficiently by deep networks, (iv) large, structured inputs like images are best processed in multiple stages, and (v) in practice, the best results for most tasks are achieved using deep networks with many layers.

Now that we understand deep and shallow network models, we turn our attention to training them. In the next chapter, we discuss loss functions. For any given parameter values $\boldsymbol{\phi}$, the loss function returns a single number that indicates the mismatch between the model outputs and the ground truth predictions for a training dataset. In chapters 6 and 7, we deal with the training process itself, in which we seek the parameter values that minimize this loss.

Notes

Deep learning: It has long been understood that it is possible to build more complex functions by composing shallow neural networks or developing networks with more than one hidden layer. Indeed, the term "deep learning" was first used by Dechter (1986). However, interest was limited due to practical concerns; it was not possible to train such networks well. The modern era of deep learning was kick-started by startling improvements in image classification reported by Krizhevsky et al. (2012). This sudden progress was arguably due to the confluence of four factors: larger training datasets, improved processing power for training, the use of the ReLU activation function, and the use of stochastic gradient descent (see chapter 6). LeCun et al. (2015) present an overview of early advances in the modern era of deep learning.

Number of linear regions: For deep networks using a total of D hidden units with ReLU activations, the upper bound on the number of regions is 2^D (Montúfar et al., 2014). The same authors show that a deep ReLU network with D_i-dimensional input and K layers, each containing $D \geq D_i$ hidden units, has $\mathcal{O}\left((D/D_i)^{(K-1)D_i} D^{D_i}\right)$ linear regions. Montúfar (2017), Arora et al. (2016) and Serra et al. (2018) all provide tighter upper bounds that consider the possibility that each layer has different numbers of hidden units. Serra et al. (2018) provide an algorithm that counts the number of linear regions in a neural network, although it is only practical for very small networks.

If the number of hidden units D in each of the K layers is the same, and D is an integer multiple of the input dimensionality D_i, then the maximum number of linear regions N_r can be

computed exactly and is:

$$N_r = \left(\frac{D}{D_i}+1\right)^{D_i(K-1)} \cdot \sum_{j=0}^{D_i} \binom{D}{j}. \tag{4.17}$$

The first term in this expression corresponds to the first $K-1$ layers of the network, which can be thought of as repeatedly folding the input space. However, we now need to devote D/D_i hidden units to each input dimension to create these folds. The last term in this equation (a sum of binomial coefficients) is the number of regions that a shallow network can create and is attributable to the last layer. For further information, consult Montúfar et al. (2014), Pascanu et al. (2013), and Montúfar (2017).

Appendix B.2 Binomial coefficient

Universal approximation theorem: We argued in section 4.5.1 that if the layers of a deep network have enough hidden units, then the width version of the universal approximation theorem applies: there exists a network that can approximate any given continuous function on a compact subset of $\mathbb{R}^{D_i}$ to arbitrary accuracy. Lu et al. (2017) proved that there exists a network with ReLU activation functions and at least D_i+4 hidden units in each layer can approximate any specified D_i-dimensional Lebesgue integrable function to arbitrary accuracy given enough layers. This is known as the *depth version* of the universal approximation theorem.

Depth efficiency: Several results show that there are functions that can be realized by deep networks but not by any shallow network whose capacity is bounded above exponentially. In other words, it would take an exponentially larger number of units in a shallow network to describe these functions accurately. This is known as the *depth efficiency* of neural networks.

Telgarsky (2016) shows that for any integer k, it is possible to construct networks with one input, one output, and $\mathcal{O}[k^3]$ layers of constant width, which cannot be realized with $\mathcal{O}[k]$ layers and less than 2^k width. Perhaps surprisingly, Eldan & Shamir (2016) showed that when there are multivariate inputs, there is a three-layer network that cannot be realized by any two-layer network if the capacity is sub-exponential in the input dimension. Cohen et al. (2016), Safran & Shamir (2017), and Poggio et al. (2017) also demonstrate functions that deep networks can approximate efficiently, but shallow ones cannot. Liang & Srikant (2016) show that for a broad class of functions, including univariate functions, shallow networks require exponentially more hidden units than deep networks for a given upper bound on the approximation error.

Width efficiency: Lu et al. (2017) investigate whether there are wide shallow networks (i.e., shallow networks with lots of hidden units) that cannot be realized by narrow networks whose depth is not substantially larger. They show that there exist classes of wide, shallow networks that can only be expressed by narrow networks with polynomial depth. This is known as the *width efficiency* of neural networks. This polynomial lower bound on width is less restrictive than the exponential lower bound on depth, suggesting that depth is more important. Vardi et al. (2022) subsequently showed that the price for making the width small is only a linear increase in the network depth for networks with ReLU activations.

Problems

Problem 4.1* Consider composing the two neural networks in figure 4.8. Draw a plot of the relationship between the input x and output y' for $x \in [-1, 1]$.

Problem 4.2 Identify the four hyperparameters in figure 4.6.

Problem 4.3 Using the non-negative homogeneity property of the ReLU function (see problem 3.5), show that:

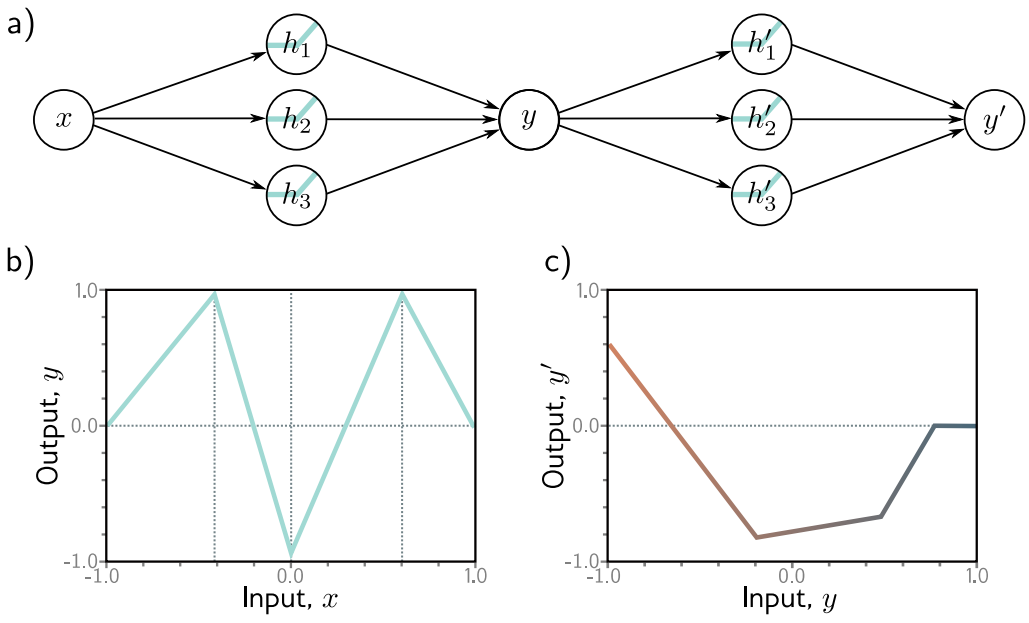

Figure 4.8 Composition of two networks for problem 4.1. a) The output y of the first network becomes the input to the second. b) The first network computes this function with output values $y \in [-1, 1]$. c) The second network computes this function on the input range $y \in [-1, 1]$.

$$\text{ReLU}\Big[\boldsymbol{\beta}_1 + \lambda_1 \cdot \boldsymbol{\Omega}_1 \text{ReLU}\left[\boldsymbol{\beta}_0 + \lambda_0 \cdot \boldsymbol{\Omega}_0 \mathbf{x}\right]\Big] = \lambda_0\lambda_1 \cdot \text{ReLU}\left[\frac{1}{\lambda_0\lambda_1}\boldsymbol{\beta}_1 + \boldsymbol{\Omega}_1 \text{ReLU}\left[\frac{1}{\lambda_0}\boldsymbol{\beta}_0 + \boldsymbol{\Omega}_0 \mathbf{x}\right]\right], \tag{4.18}$$

where λ_0 and λ_1 are non-negative scalars. From this, we see that the weight matrices can be rescaled by any magnitude as long as the biases are also adjusted, and the scale factors can be re-applied at the end of the network.

Problem 4.4 Write out the equations for a deep neural network that takes $D_i = 5$ inputs, $D_o = 4$ outputs and has three hidden layers of sizes $D_1 = 20$, $D_2 = 10$, and $D_3 = 7$, respectively, in both the forms of equations 4.15 and 4.16. What are the sizes of each weight matrix $\boldsymbol{\Omega}_\bullet$ and bias vector $\boldsymbol{\beta}_\bullet$?

Problem 4.5 Consider a deep neural network with $D_i = 5$ inputs, $D_o = 1$ output, and $K = 20$ hidden layers containing $D = 30$ hidden units each. What is the depth of this network? What is the width?

Problem 4.6 Consider a network with $D_i = 1$ input, $D_o = 1$ output, and $K = 10$ layers, with $D = 10$ hidden units in each. Would the number of weights increase more if we increased the depth by one or the width by one? Provide your reasoning.

Problem 4.7 Choose values for the parameters $\boldsymbol{\phi} = \{\phi_0, \phi_1, \phi_2, \phi_3, \theta_{10}, \theta_{11}, \theta_{20}, \theta_{21}, \theta_{30}, \theta_{31}\}$ for the shallow neural network in equation 3.1 (with ReLU activation functions) that will define an identity function over a finite range $x \in [a, b]$.

Problem 4.8* Figure 4.9 shows the activations in the three hidden units of a shallow network (as in figure 3.3). The slopes in the hidden units are 1.0, 1.0, and -1.0, respectively, and the "joints" in the hidden units are at positions 1/6, 2/6, and 4/6. Find values of ϕ_0, ϕ_1, ϕ_2, and ϕ_3 that will combine the hidden unit activations as $\phi_0 + \phi_1 h_1 + \phi_2 h_2 + \phi_3 h_3$ to create a function with four linear regions that oscillate between output values of zero and one. The slope of the leftmost region should be positive, the next one negative, and so on. How many linear regions will we create if we compose this network with itself? How many will we create if we compose it with itself K times?

Problem 4.9* Following problem 4.8, is it possible to create a function with three linear regions that oscillates back and forth between output values of zero and one using a shallow network with two hidden units? Is it possible to create a function with five linear regions that oscillates in the same way using a shallow network with four hidden units?

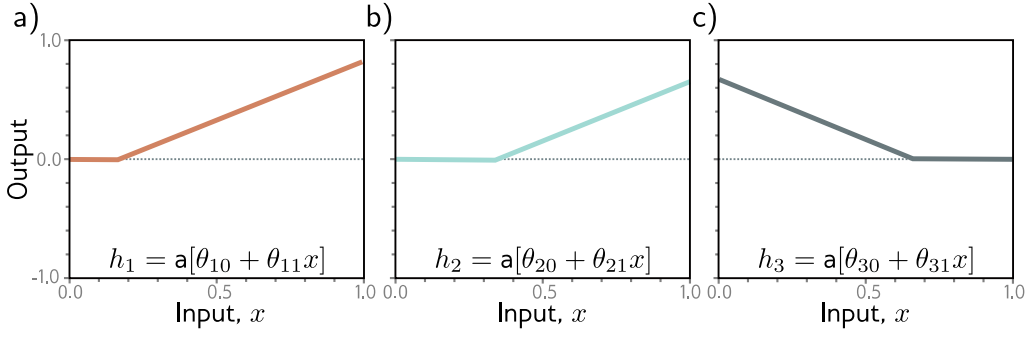

Figure 4.9 Hidden unit activations for problem 4.8. a) First hidden unit has a joint at position $x = 1/6$ and a slope of one in the active region. b) Second hidden unit has a joint at position $x = 2/6$ and a slope of one in the active region. c) Third hidden unit has a joint at position $x = 4/6$ and a slope of minus one in the active region.

Problem 4.10 Consider a deep neural network with a single input, a single output, and K hidden layers, each of which contains D hidden units. Show that this network will have a total of $3D + 1 + (K - 1)D(D + 1)$ parameters.

Problem 4.11* Consider two neural networks that map a scalar input x to a scalar output y. The first network is shallow and has $D = 95$ hidden units. The second is deep and has $K = 10$ layers, each containing $D = 5$ hidden units. How many parameters does each network have? How many linear regions can each network make (see equation 4.17)? Which would run faster?