Chapter 3

Shallow neural networks

Chapter 2 introduced supervised learning using 1D linear regression. However, this model can only describe the input/output relationship as a line. This chapter introduces shallow neural networks. These describe piecewise linear functions and are expressive enough to approximate arbitrarily complex relationships between multi-dimensional inputs and outputs.

3.1 Neural network example

Shallow neural networks are functions $\mathbf{y} = \mathbf{f}[\mathbf{x}, \boldsymbol{\phi}]$ with parameters $\boldsymbol{\phi}$ that map multivariate inputs $\mathbf{x}$ to multivariate outputs $\mathbf{y}$. We defer a full definition until section 3.4 and introduce the main ideas using an example network $\mathrm{f}[x, \boldsymbol{\phi}]$ that maps a scalar input x to a scalar output y and has ten parameters $\boldsymbol{\phi} = \{\phi_0, \phi_1, \phi_2, \phi_3, \theta_{10}, \theta_{11}, \theta_{20}, \theta_{21}, \theta_{30}, \theta_{31}\}$:

$$\begin{aligned} y &= \mathrm{f}[x, \boldsymbol{\phi}] \\ &= \phi_0 + \phi_1 \mathrm{a}[\theta_{10} + \theta_{11}x] + \phi_2 \mathrm{a}[\theta_{20} + \theta_{21}x] + \phi_3 \mathrm{a}[\theta_{30} + \theta_{31}x]. \end{aligned} \tag{3.1}$$

We can break down this calculation into three parts: first, we compute three linear functions of the input data ($\theta_{10} + \theta_{11}x$, $\theta_{20} + \theta_{21}x$, and $\theta_{30} + \theta_{31}x$). Second, we pass the three results through an *activation function* $\mathrm{a}[\bullet]$. Finally, we weight the three resulting activations with ϕ_1, ϕ_2, and ϕ_3, sum them, and add an offset ϕ_0.

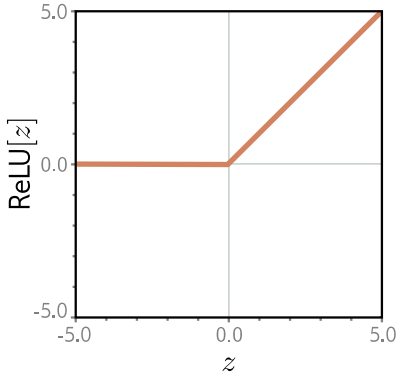
To complete the description, we must define the activation function $\mathrm{a}[\bullet]$. There are many possibilities, but the most common choice is the *rectified linear unit* or *ReLU*:

$$\mathrm{a}[z] = \mathrm{ReLU}[z] = \begin{cases} 0 & z < 0 \\ z & z \geq 0 \end{cases}. \tag{3.2}$$

This returns the input when it is positive and zero otherwise (figure 3.1).

It is probably not obvious which family of input/output relations is represented by equation 3.1. Nonetheless, the ideas from the previous chapter are all applicable. Equation 3.1 represents a family of functions where the particular member of the family

Figure 3.1 Rectified linear unit (ReLU). This activation function returns zero if the input is less than zero and returns the input unchanged otherwise. In other words, it clips negative values to zero. Note that there are many other possible choices for the activation function (see figure 3.13), but the ReLU is the most commonly used and the easiest to understand.



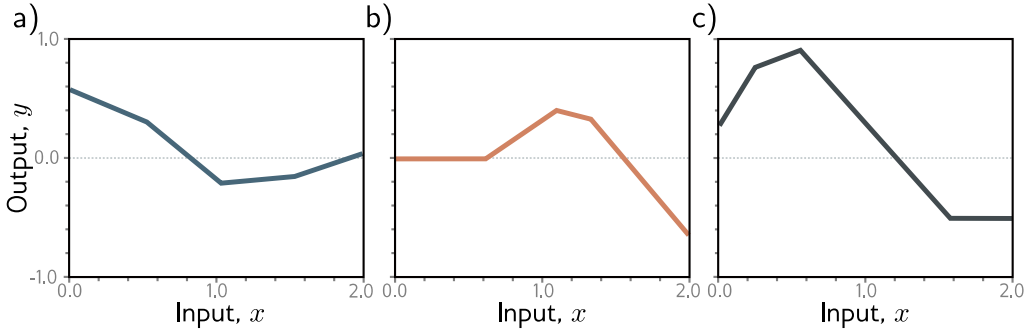


Figure 3.2 Family of functions defined by equation 3.1. a–c) Functions for three different choices of the ten parameters $\boldsymbol{\phi}$. In each case, the input/output relation is piecewise linear. However, the positions of the joints, the slopes of the linear regions between them, and the overall height vary.

depends on the ten parameters in $\boldsymbol{\phi}$. If we know these parameters, we can perform inference (predict y) by evaluating the equation for a given input x. Given a training dataset $\{x_i, y_i\}_{i=1}^{I}$, we can define a least squares loss function $L[\boldsymbol{\phi}]$ and use this to measure how effectively the model describes this dataset for any given parameter values $\boldsymbol{\phi}$. To train the model, we search for the values $\hat{\boldsymbol{\phi}}$ that minimize this loss.

3.1.1 Neural network intuition

In fact, equation 3.1 represents a family of continuous piecewise linear functions (figure 3.2) with up to four linear regions. We now break down equation 3.1 and show *why* it describes this family. To make this easier to understand, we split the function into two parts. First, we introduce the intermediate quantities:

$$\begin{aligned} h_1 &= \text{a}[\theta_{10} + \theta_{11}x] \\ h_2 &= \text{a}[\theta_{20} + \theta_{21}x] \\ h_3 &= \text{a}[\theta_{30} + \theta_{31}x], \end{aligned} \tag{3.3}$$

where we refer to h_1, h_2, and h_3 as *hidden units*. Second, we compute the output by combining these hidden units with a linear function:[1]

$$y = \phi_0 + \phi_1 h_1 + \phi_2 h_2 + \phi_3 h_3. \tag{3.4}$$

Figure 3.3 shows the flow of computation that creates the function in figure 3.2a. Each hidden unit contains a linear function $\theta_{\bullet 0} + \theta_{\bullet 1}x$ of the input, and that line is clipped by the ReLU function $a[\bullet]$ below zero. The positions where the three lines cross zero become the three "joints" in the final output. The three clipped lines are then weighted by ϕ_1, ϕ_2, and ϕ_3, respectively. Finally, the offset ϕ_0 is added, which controls the overall height of the final function.

Problems 3.1–3.8

Each linear region in figure 3.3j corresponds to a different *activation pattern* in the hidden units. When a unit is clipped, we refer to it as *inactive*, and when it is not clipped, we refer to it as *active*. For example, the shaded region receives contributions from h_1 and h_3 (which are active) but not from h_2 (which is inactive). The slope of each linear region is determined by the original slopes $\theta_{\bullet 1}$ of the active inputs for this region and the weights $\phi_\bullet$ that were subsequently applied. For example, the slope in the shaded region (see problem 3.3) is $\theta_{11}\phi_1 + \theta_{31}\phi_3$, where the first term is the slope in panel (g) and the second term is the slope in panel (i).

Each hidden unit contributes one "joint" to the function, so with three hidden units, there can be four linear regions. However, only three of the slopes of these regions are independent; the fourth is either zero (if all the hidden units are inactive in this region) or is a sum of slopes from the other regions.

Notebook 3.1
Shallow networks I

Problem 3.9

3.1.2 Depicting neural networks

We have been discussing a neural network with one input, one output, and three hidden units. We visualize this network in figure 3.4a. The input is on the left, the hidden units are in the middle, and the output is on the right. Each connection represents one of the ten parameters. To simplify this representation, we do not typically draw the intercept parameters, so this network is usually depicted as in figure 3.4b.

[1] For the purposes of this book, a linear function has the form $z' = \phi_0 + \sum_i \phi_i z_i$. Any other type of function is nonlinear. For instance, the ReLU function (equation 3.2) and the example neural network that contains it (equation 3.1) are both nonlinear. See notes at end of chapter for further clarification.

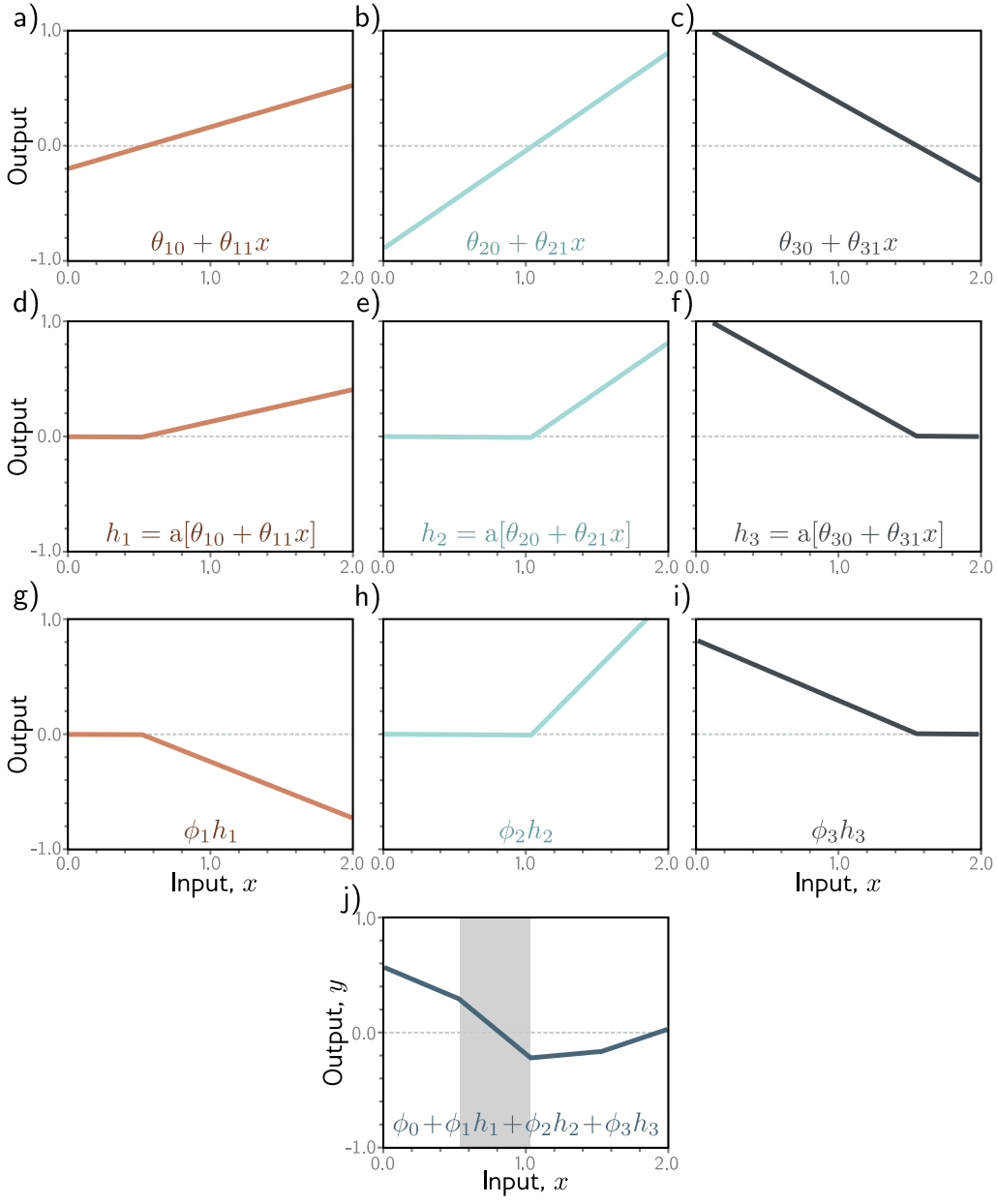


Figure 3.3 Computation for function in figure 3.2a. a–c) The input x is passed through three linear functions, each with a different y-intercept $\theta_{\bullet 0}$ and slope $\theta_{\bullet 1}$. d–f) Each line is passed through the ReLU activation function, which clips negative values to zero. g–i) The three clipped lines are then weighted (scaled) by 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 height is added. Each of the four linear regions corresponds to a different activation pattern in the hidden units. In the shaded region, h_2 is inactive (clipped), but h_1 and h_3 are both active.

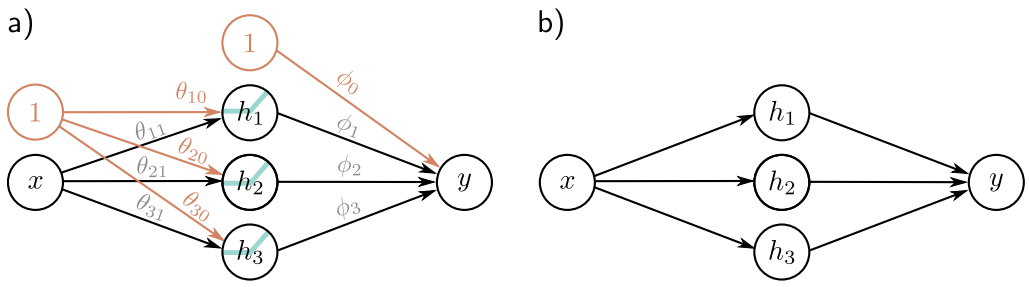


Figure 3.4 Depicting neural networks. a) The input x is on the left, the hidden units h_1, h_2, and h_3 in the center, and the output y on the right. Computation flows from left to right. The input is used to compute the hidden units, which are combined to create the output. Each of the ten arrows represents a parameter (intercepts in orange and slopes in black). Each parameter multiplies its source and adds the result to its target. For example, we multiply the parameter ϕ_1 by source h_1 and add it to y. We introduce additional nodes containing ones (orange circles) to incorporate the offsets into this scheme, so we multiply ϕ_0 by one (with no effect) and add it to y. ReLU functions are applied at the hidden units. b) More typically, the intercepts, ReLU functions, and parameter names are omitted; this simpler depiction represents the same network.

3.2 Universal approximation theorem

In the previous section, we introduced an example neural network with one input, one output, ReLU activation functions, and three hidden units. Let's now generalize this slightly and consider the case with D hidden units where the d^{th} hidden unit is:

$$h_d = \mathrm{a}[\theta_{d0} + \theta_{d1}x], \tag{3.5}$$

and these are combined linearly to create the output:

$$y = \phi_0 + \sum_{d=1}^{D} \phi_d h_d. \tag{3.6}$$

The number of hidden units in a shallow network is a measure of the *network capacity*. With ReLU activation functions, the output of a network with D hidden units has at most D joints and so is a piecewise linear function with at most $D+1$ linear regions. As we add more hidden units, the model can approximate more complex functions.

Problem 3.10

Indeed, with enough capacity (hidden units), a shallow network can describe any continuous 1D function defined on a compact subset of the real line to arbitrary precision. To see this, consider that every time we add a hidden unit, we add another linear region to the function. As these regions become more numerous, they represent smaller sections of the function, which are increasingly well approximated by a line (figure 3.5). The *universal approximation theorem* proves that for any continuous function, there exists a shallow network that can approximate this function to any specified precision.

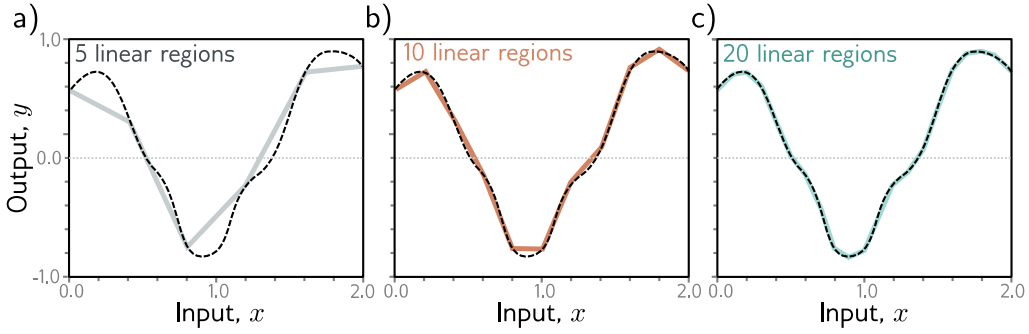


Figure 3.5 Approximation of a 1D function (dashed line) by a piecewise linear model. a–c) As the number of regions increases, the model becomes closer and closer to the continuous function. A neural network with a scalar input creates one extra linear region per hidden unit. The universal approximation theorem proves that, with enough hidden units, there exists a shallow neural network that can describe any given continuous function defined on a compact subset of $\mathbb{R}^{D_i}$ to arbitrary precision.

3.3 Multivariate inputs and outputs

In the above example, the network has a single scalar input x and a single scalar output y. However, the universal approximation theorem also holds for the more general case where the network maps multivariate inputs $\mathbf{x} = [x_1, x_2, \ldots, x_{D_i}]^T$ to multivariate output predictions $\mathbf{y} = [y_1, y_2, \ldots, y_{D_o}]^T$. We first explore how to extend the model to predict multivariate outputs. Then we consider multivariate inputs. Finally, in section 3.4, we present a general definition of a shallow neural network.

3.3.1 Visualizing multivariate outputs

To extend the network to multivariate outputs $\mathbf{y}$, we simply use a different linear function of the hidden units for each output. So, a network with a scalar input x, four hidden units h_1, h_2, h_3, and h_4, and a 2D multivariate output $\mathbf{y} = [y_1, y_2]^T$ would be defined as:

$$\begin{aligned} h_1 &= \mathrm{a}[\theta_{10} + \theta_{11}x] \\ h_2 &= \mathrm{a}[\theta_{20} + \theta_{21}x] \\ h_3 &= \mathrm{a}[\theta_{30} + \theta_{31}x] \\ h_4 &= \mathrm{a}[\theta_{40} + \theta_{41}x], \end{aligned} \tag{3.7}$$

and

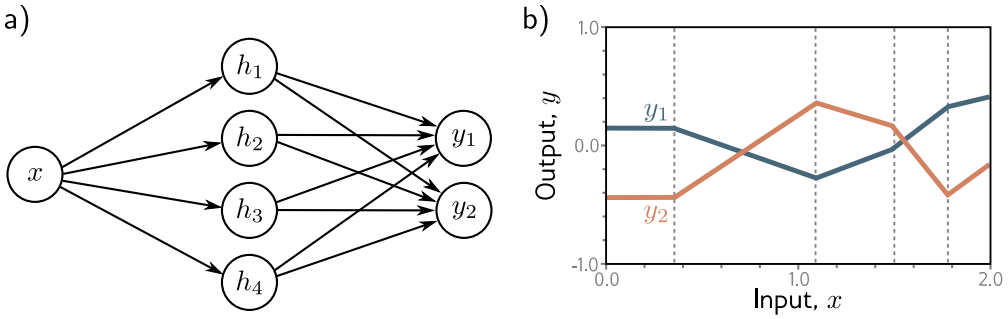


Figure 3.6 Network with one input, four hidden units, and two outputs. a) Visualization of network structure. b) This network produces two piecewise linear functions, $y_1[x]$ and $y_2[x]$. The four "joints" of these functions (at vertical dotted lines) are constrained to be in the same places since they share the same hidden units, but the slopes and overall height may differ.

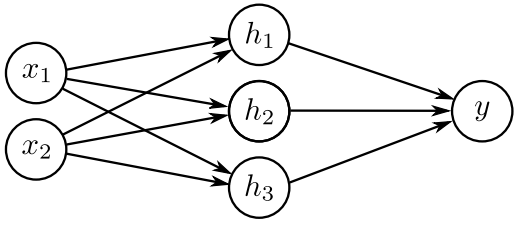


Figure 3.7 Visualization of neural network with 2D multivariate input $\mathbf{x} = [x_1, x_2]^T$ and scalar output y.

$$\begin{aligned} y_1 &= \phi_{10} + \phi_{11}h_1 + \phi_{12}h_2 + \phi_{13}h_3 + \phi_{14}h_4 \\ y_2 &= \phi_{20} + \phi_{21}h_1 + \phi_{22}h_2 + \phi_{23}h_3 + \phi_{24}h_4. \end{aligned} \tag{3.8}$$

The two outputs are two different linear functions of the hidden units.

As we saw in figure 3.3, the "joints" in the piecewise functions depend on where the initial linear functions $\theta_{\bullet 0} + \theta_{\bullet 1}x$ are clipped by the ReLU functions $a[\bullet]$ at the hidden units. Since both outputs y_1 and y_2 are different linear functions of the same four hidden units, the four "joints" in each must be in the same places. However, the slopes of the linear regions and the overall vertical offset can differ (figure 3.6). Problem 3.11

3.3.2 Visualizing multivariate inputs

To cope with multivariate inputs $\mathbf{x}$, we extend the linear relations between the input and the hidden units. So a network with two inputs $\mathbf{x} = [x_1, x_2]^T$ and a scalar output y (figure 3.7) might have three hidden units defined by:

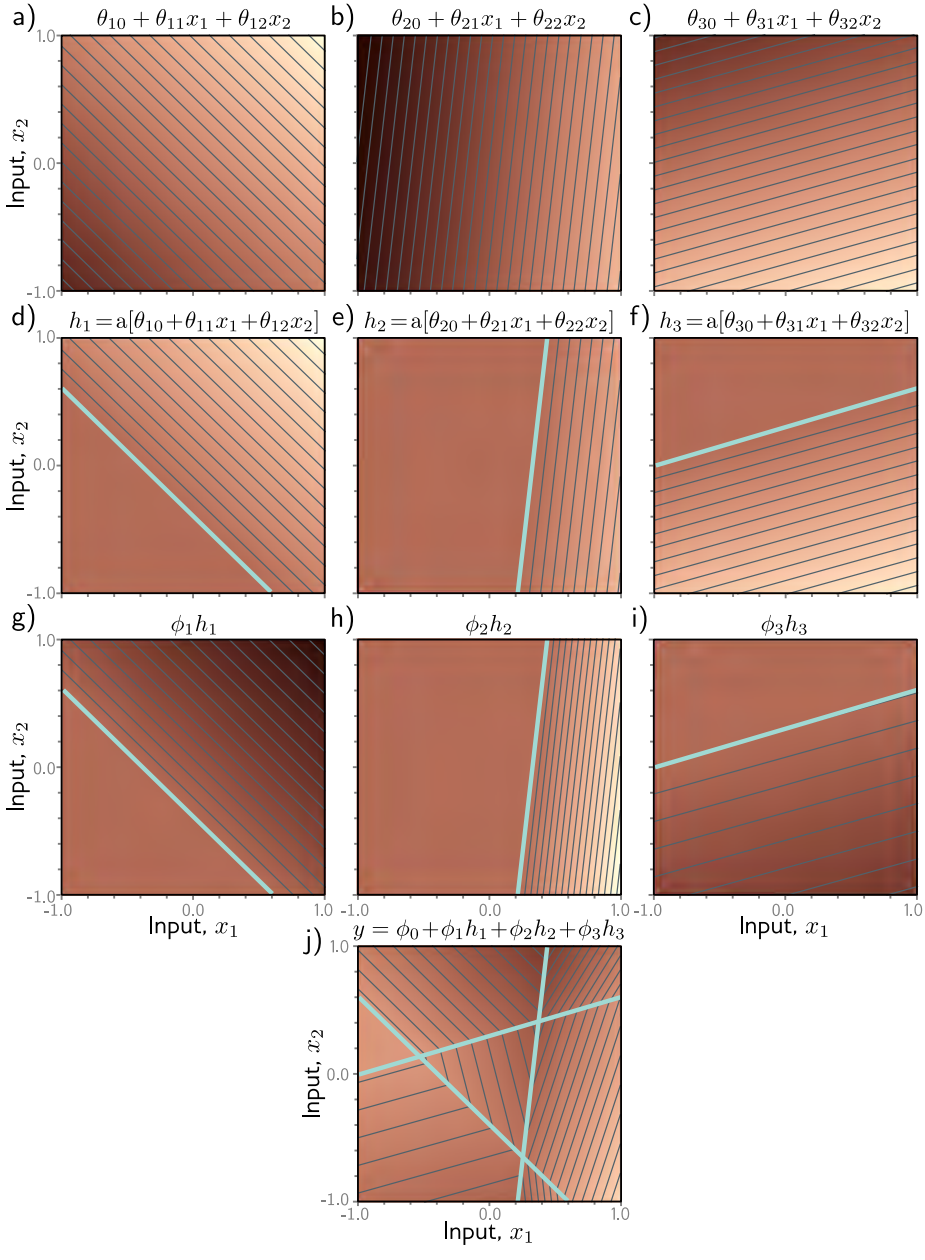


Figure 3.8 Processing in network with two inputs $\mathbf{x} = [x_1, x_2]^T$, three hidden units h_1, h_2, h_3, and one output y. a–c) The input to each hidden unit is a linear function of the two inputs, which corresponds to an oriented plane. Brightness indicates function output. For example, in panel (a), the brightness represents $\theta_{10} + \theta_{11}x_1 + \theta_{12}x_2$. Thin lines are contours. d–f) Each plane is clipped by the ReLU activation function (cyan lines are equivalent to "joints" in figures 3.3d–f). g-i) The clipped planes are then weighted, and j) summed together with an offset that determines the overall height of the surface. The result is a continuous surface made up of convex piecewise linear polygonal regions.

$$\begin{aligned} h_1 &= \mathrm{a}[\theta_{10} + \theta_{11}x_1 + \theta_{12}x_2] \\ h_2 &= \mathrm{a}[\theta_{20} + \theta_{21}x_1 + \theta_{22}x_2] \\ h_3 &= \mathrm{a}[\theta_{30} + \theta_{31}x_1 + \theta_{32}x_2], \end{aligned} \tag{3.9}$$

where there is now one slope parameter for each input. The hidden units are combined to form the output in the usual way:

$$y = \phi_0 + \phi_1 h_1 + \phi_2 h_2 + \phi_3 h_3. \tag{3.10}$$

Figure 3.8 illustrates the processing of this network. Each hidden unit receives a linear combination of the two inputs, which forms an oriented plane in the 3D input/output space. The activation function clips the negative values of these planes to zero. The clipped planes are then recombined in a second linear function (equation 3.10) to create a continuous piecewise linear surface consisting of convex polygonal regions (figure 3.8j). Each region corresponds to a different activation pattern. For example, in the central triangular region, the first and third hidden units are active, and the second is inactive.

Problems 3.12–3.13

Notebook 3.2 Shallow networks II

Appendix B.1.2 Convex region

When there are more than two inputs to the model, it becomes difficult to visualize. However, the interpretation is similar. The output will be a continuous piecewise linear function of the input, where the linear regions are now convex polytopes in the multi-dimensional input space.

Note that as the input dimensions grow, the number of linear regions increases rapidly (figure 3.9). To get a feeling for how rapidly, consider that each hidden unit defines a hyperplane that delineates the part of space where this unit is active from the part where it is not (cyan lines in 3.8d–f). If we had the same number of hidden units as input dimensions D_i, we could align each hyperplane with one of the coordinate axes (figure 3.10). For two input dimensions, this would divide the space into four quadrants. For three dimensions, this would create eight octants, and for D_i dimensions, this would create 2^{D_i} orthants. Shallow neural networks usually have more hidden units than input dimensions, so they typically create more than 2^{D_i} linear regions.

Notebook 3.3 Shallow network regions

3.4 Shallow neural networks: general case

We have described several example shallow networks to help develop intuition about how they work. We now define a general equation for a shallow neural network $\mathbf{y} = \mathbf{f}[\mathbf{x}, \boldsymbol{\phi}]$ that maps a multi-dimensional input $\mathbf{x} \in \mathbb{R}^{D_i}$ to a multi-dimensional output $\mathbf{y} \in \mathbb{R}^{D_o}$ using $\mathbf{h} \in \mathbb{R}^{D}$ hidden units. Each hidden unit is computed as:

$$h_d = \mathrm{a}\left[\theta_{d0} + \sum_{i=1}^{D_i} \theta_{di} x_i\right], \tag{3.11}$$

and these are combined linearly to create the output:

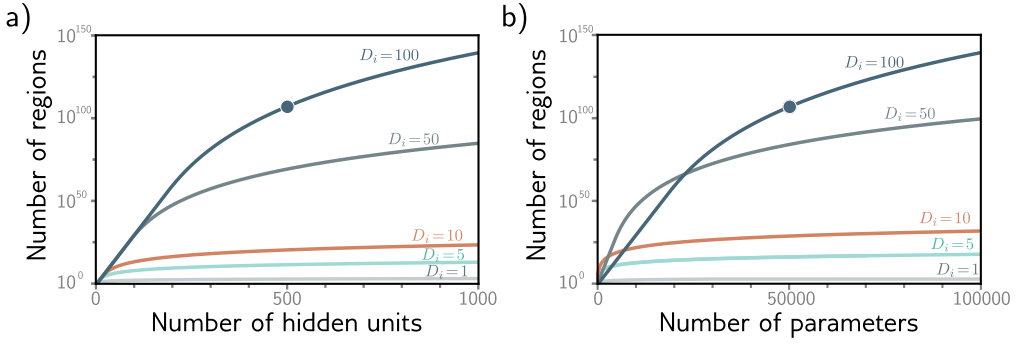


Figure 3.9 Linear regions vs. hidden units. a) Maximum possible regions as a function of the number of hidden units for five different input dimensions $D_i = \{1, 5, 10, 50, 100\}$. The number of regions increases rapidly in high dimensions; with $D = 500$ units and input size $D_i = 100$, there can be greater than 10^{107} regions (solid circle). b) The same data are plotted as a function of the number of parameters. The solid circle represents the same model as in panel (a) with $D = 500$ hidden units. This network has $51,001$ parameters and would be considered very small by modern standards.

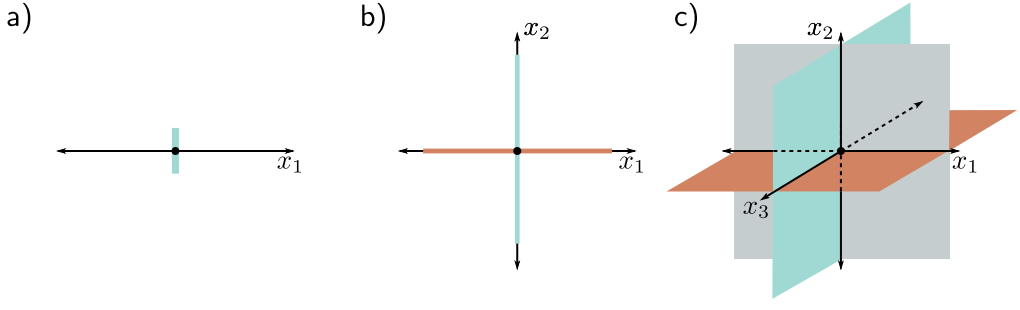


Figure 3.10 Number of linear regions vs. input dimensions. a) With a single input dimension, a model with one hidden unit creates one joint, which divides the axis into two linear regions. b) With two input dimensions, a model with two hidden units can divide the input space using two lines (here aligned with axes) to create four regions. c) With three input dimensions, a model with three hidden units can divide the input space using three planes (again aligned with axes) to create eight regions. Continuing this argument, it follows that a model with D_i input dimensions and D_i hidden units can divide the input space with D_i hyperplanes to create 2^{D_i} linear regions.

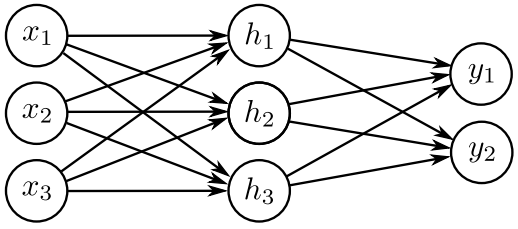


Figure 3.11 Visualization of neural network with three inputs and two outputs. This network has twenty parameters. There are fifteen slopes (indicated by arrows) and five offsets (not shown).

$$y_j = \phi_{j0} + \sum_{d=1}^{D} \phi_{jd} h_d, \tag{3.12}$$

where $a[\bullet]$ is a nonlinear activation function. The model has parameters $\boldsymbol{\phi} = \{\theta_{\bullet\bullet}, \phi_{\bullet\bullet}\}$. Figure 3.11 shows an example with three inputs, three hidden units, and two outputs.

Problems 3.14–3.17

The activation function permits the model to describe nonlinear relations between input and the output, and as such, it must be nonlinear itself; with no activation function, or a linear activation function, the overall mapping from input to output would be restricted to be linear. Many different activation functions have been tried (see figure 3.13), but the most common choice is the ReLU (figure 3.1), which has the merit of being easily interpretable. With ReLU activations, the network divides the input space into convex polytopes defined by the intersections of hyperplanes computed by the "joints" in the ReLU functions. Each convex polytope contains a different linear function. The polytopes are the same for each output, but the linear functions they contain can differ.

Notebook 3.4 Activation functions

3.5 Terminology

We conclude this chapter by introducing some terminology. Regrettably, neural networks have a lot of associated jargon. They are often referred to in terms of *layers*. The left of figure 3.12 is the *input layer*, the center is the *hidden layer*, and to the right is the *output layer*. We would say that the network in figure 3.12 has one hidden layer containing four hidden units. The hidden units themselves are sometimes referred to as *neurons*. When we pass data through the network, the values of the inputs to the hidden layer (i.e., before the ReLU functions are applied) are termed *pre-activations*. The values at the hidden layer (i.e., after the ReLU functions) are termed *activations*.

For historical reasons, any neural network with at least one hidden layer is also called a *multi-layer perceptron*, or *MLP* for short. Networks with one hidden layer (as described in this chapter) are sometimes referred to as *shallow neural networks*. Networks with multiple hidden layers (as described in the next chapter) are referred to as *deep neural networks*. Neural networks in which the connections form an acyclic graph (i.e., a graph with no loops, as in all the examples in this chapter) are referred to as *feed-forward networks*. If every element in one layer connects to every element in the next (as in all the examples in this chapter), the network is *fully connected*. These connections

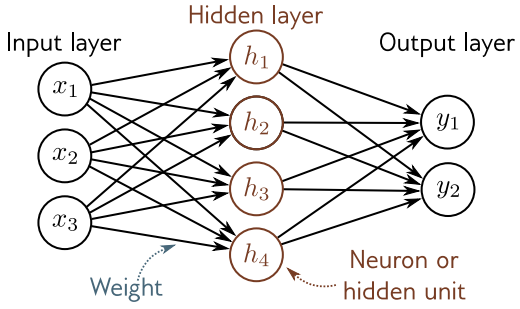


Figure 3.12 Terminology. A shallow network consists of an input layer, a hidden layer, and an output layer. Each layer is connected to the next by forward connections (arrows). For this reason, these models are referred to as feed-forward networks. When every variable in one layer connects to every variable in the next, we call this a fully connected network. Each connection represents a slope parameter in the underlying equation, and these parameters are termed weights. The variables in the hidden layer are termed neurons or hidden units. The values feeding into the hidden units are termed pre-activations, and the values at the hidden units (i.e., after the ReLU function is applied) are termed activations.

represent slope parameters in the underlying equations and are referred to as *network weights*. The offset parameters (not shown in figure 3.12) are called *biases*.

3.6 Summary

Shallow neural networks have one hidden layer. They (i) compute several linear functions of the input, (ii) pass each result through an activation function, and then (iii) take a linear combination of these activations to form the outputs. Shallow neural networks make predictions $\mathbf{y}$ based on inputs $\mathbf{x}$ by dividing the input space into a continuous surface of piecewise linear regions. With enough hidden units (neurons), shallow neural networks can approximate any continuous function to arbitrary precision.

Chapter 4 discusses deep neural networks, which extend the models from this chapter by adding more hidden layers. Chapters 5–7 describe how to train these models.

Notes

"Neural" networks: If the models in this chapter are just functions, why are they called "neural networks"? The connection is, unfortunately, tenuous. Visualizations like figure 3.12 consist of nodes (inputs, hidden units, and outputs) that are densely connected to one another. This bears a superficial similarity to neurons in the mammalian brain, which also have dense connections. However, there is scant evidence that brain computation works in the same way as neural networks, and it is unhelpful to think about biology going forward.

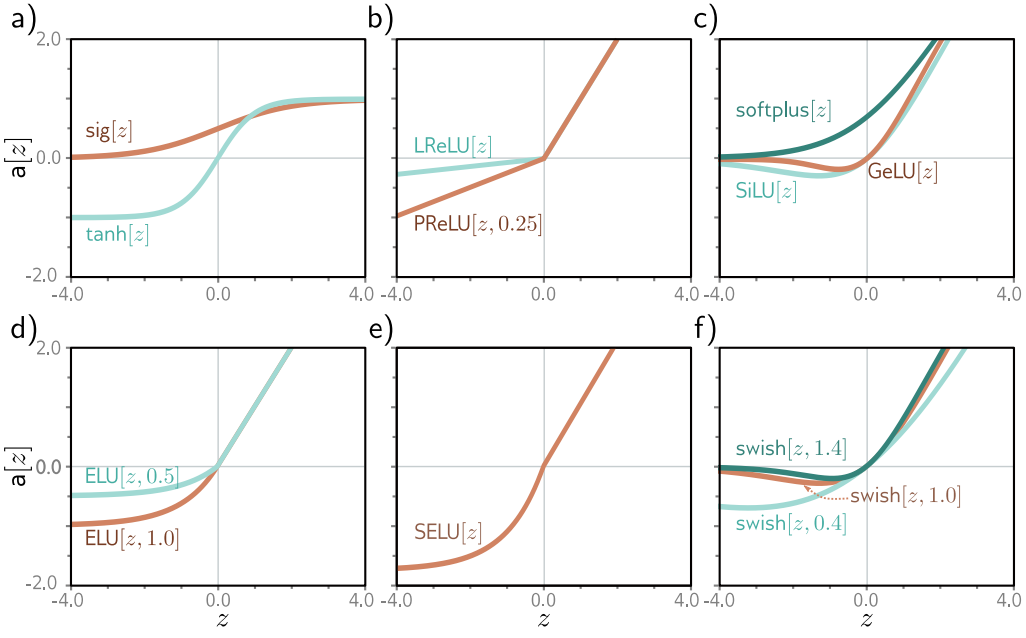


Figure 3.13 Activation functions. a) Logistic sigmoid and tanh functions. b) Leaky ReLU and parametric ReLU with parameter 0.25. c) SoftPlus, Gaussian error linear unit, and sigmoid linear unit. d) Exponential linear unit with parameters 0.5 and 1.0, e) Scaled exponential linear unit. f) Swish with parameters 0.4, 1.0, and 1.4.

History of neural networks: McCulloch & Pitts (1943) first came up with the notion of an artificial neuron that combined inputs to produce an output, but this model did not have a practical learning algorithm. Rosenblatt (1958) developed the *perceptron*, which linearly combined inputs and then thresholded them to make a yes/no decision. He also provided an algorithm to learn the weights from data. Minsky & Papert (1969) argued that the linear function was inadequate for general classification problems but that adding hidden layers with nonlinear activation functions (hence the term multi-layer perceptron) could allow the learning of more general input/output relations. However, they concluded that Rosenblatt's algorithm could not learn the parameters of such models. It was not until the 1980s that a practical algorithm (backpropagation, see chapter 7) was developed, and significant work on neural networks resumed. The history of neural networks is chronicled by Kurenkov (2020), Sejnowski (2018), and Schmidhuber (2022).

Activation functions: The ReLU function has been used as far back as Fukushima (1969). However, in the early days of neural networks, it was more common to use the logistic sigmoid or tanh activation functions (figure 3.13a). The ReLU was re-popularized by Jarrett et al. (2009), Nair & Hinton (2010), and Glorot et al. (2011) and is an important part of the success story of modern neural networks. It has the nice property that the derivative of the output with respect to the input is always one for inputs greater than zero. This contributes to the stability and efficiency of training (see chapter 7) and contrasts with the derivatives of sigmoid activation

functions, which saturate (become close to zero) for large positive and large negative inputs.

However, the ReLU function has the disadvantage that its derivative is zero for negative inputs. If all the training examples produce negative inputs to a given ReLU function, then we cannot improve the parameters feeding into this ReLU during training. The gradient with respect to the incoming weights is locally flat, so we cannot "walk downhill." This is known as the *dying ReLU* problem. Many variations on the ReLU have been proposed to resolve this problem (figure 3.13b), including (i) the leaky ReLU (Maas et al., 2013), which also has a linear output for negative values with a smaller slope of 0.1, (ii) the parametric ReLU (He et al., 2015), which treats the slope of the negative portion as an unknown parameter, and (iii) the concatenated ReLU (Shang et al., 2016), which produces two outputs, one of which clips below zero (i.e., like a typical ReLU) and one of which clips above zero.

A variety of smooth functions have also been investigated (figure 3.13c–d), including the softplus function (Glorot et al., 2011), Gaussian error linear unit (Hendrycks & Gimpel, 2016), sigmoid linear unit (Hendrycks & Gimpel, 2016), and exponential linear unit (Clevert et al., 2015). Most of these are attempts to avoid the dying ReLU problem while limiting the gradient for negative values. Klambauer et al. (2017) introduced the scaled exponential linear unit (figure 3.13e), which is particularly interesting as it helps stabilize the variance of the activations when the input variance has a limited range (see section 7.5). Ramachandran et al. (2017) adopted an empirical approach to choosing an activation function. They searched the space of possible functions to find the one that performed best over a variety of supervised learning tasks. The optimal function was found to be $a[x] = x/(1+\exp[-\beta x])$, where β is a learned parameter (figure 3.13f). They termed this function *Swish*. Interestingly, this was a rediscovery of activation functions previously proposed by Hendrycks & Gimpel (2016) and Elfwing et al. (2018). Howard et al. (2019) approximated Swish by the HardSwish function, which has a very similar shape but is faster to compute:

$$\text{HardSwish}[z] = \begin{cases} 0 & z < -3 \\ z(z+3)/6 & -3 \leq z \leq 3 \\ z & z > 3 \end{cases}. \tag{3.13}$$

There is no definitive answer as to which of these activations functions is empirically superior. However, the leaky ReLU, parameterized ReLU, and many of the continuous functions can be shown to provide minor performance gains over the ReLU in particular situations. We restrict attention to neural networks with the basic ReLU function for the rest of this book because it's easy to characterize the functions they create in terms of the number of linear regions.

Universal approximation theorem: The *width version* of this theorem states that there exists a network with one hidden layer containing a finite number of hidden units that can approximate any specified continuous function on a compact subset of $\mathbb{R}^n$ to arbitrary accuracy. This was proved by Cybenko (1989) for a class of sigmoid activations and was later shown to be true for a larger class of nonlinear activation functions (Hornik, 1991).

Number of linear regions: Consider a shallow network with $D_i \geq 2$-dimensional inputs and D hidden units. The number of linear regions is determined by the intersections of the D hyperplanes created by the "joints" in the ReLU functions (e.g., figure 3.8d–f). Each region is created by a different combination of the ReLU functions clipping or not clipping the input. The number of regions created by D hyperplanes in the $D_i \leq D$-dimensional input space was shown by Zaslavsky (1975) to be at most $\sum_{j=0}^{D_i} \binom{D}{j}$ (i.e., a sum of binomial coefficients). As a rule of thumb, shallow neural networks almost always have a larger number D of hidden units than input dimensions D_i and create between 2^{D_i} and 2^D linear regions.

Appendix B.2 Binomial coefficient

Problem 3.18

Linear, affine, and nonlinear functions: Technically, a linear transformation f[•] is any function that obeys the principle of superposition, so $f[a+b] = f[a]+f[b]$. This definition implies that $f[2a] = 2f[a]$.The weighted sum $f[h_1, h_2, h_3] = \phi_1 h_1 + \phi_2 h_2 + \phi_3 h_3$ is linear, but once the offset (bias) is added so $f[h_1, h_2, h_3] = \phi_0 + \phi_1 h_1 + \phi_2 h_2 + \phi_3 h_3$, this is no longer true. To see this, consider that the output is doubled when we double the arguments of the former function. This is not the case for the latter function, which is more properly termed an *affine* function. However, it is common in machine learning to conflate these terms. We follow this convention in this book and refer to both as linear. All other functions we will encounter are nonlinear.

Problems

Problem 3.1 What kind of mapping from input to output would be created if the activation function in equation 3.1 was linear so that $a[z] = \psi_0 + \psi_1 z$? What kind of mapping would be created if the activation function was removed, so $a[z] = z$?

Problem 3.2 For each of the four linear regions in figure 3.3j, indicate which hidden units are inactive and which are active (i.e., which do and do not clip their inputs).

Problem 3.3* Derive expressions for the positions of the "joints" in function in figure 3.3j in terms of the ten parameters $\boldsymbol{\phi}$ and the input x. Derive expressions for the slopes of the four linear regions.

Problem 3.4 Draw a version of figure 3.3 where the y-intercept and slope of the third hidden unit have changed as in figure 3.14c. Assume that the remaining parameters remain the same.

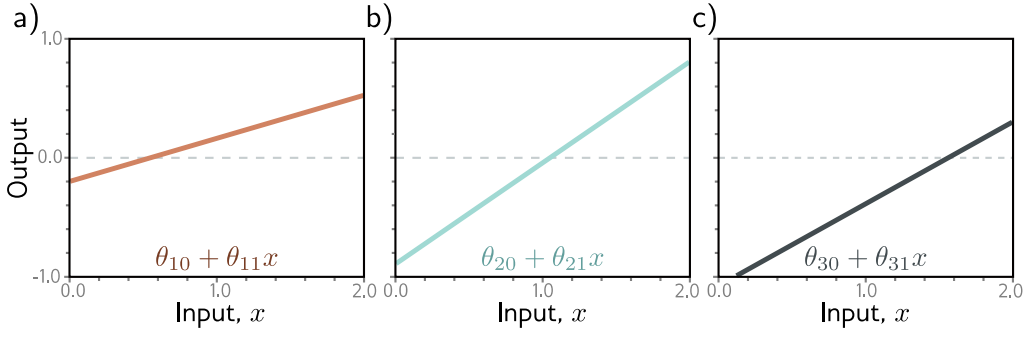


Figure 3.14 Processing in network with one input, three hidden units, and one output for problem 3.4. a–c) The input to each hidden unit is a linear function of the inputs. The first two are the same as in figure 3.3, but the last one differs.

Problem 3.5 Prove that the following property holds for $\alpha \in \mathbb{R}^+$:

$$\text{ReLU}[\alpha \cdot z] = \alpha \cdot \text{ReLU}[z]. \tag{3.14}$$

This is known as the *non-negative homogeneity* property of the ReLU function.

Problem 3.6 Following on from problem 3.5, what happens to the shallow network defined in equations 3.3 and 3.4 when we multiply the parameters θ_{10} and θ_{11} by a positive constant α and divide the slope ϕ_1 by the same parameter α? What happens if α is negative?

Problem 3.7 Consider fitting the model in equation 3.1 using a least squares loss function. Does this loss function have a unique minimum? i.e., is there a single "best" set of parameters?

Problem 3.8 Consider replacing the ReLU activation function with (i) the Heaviside step function heaviside$[z]$, (ii) the hyperbolic tangent function $\tanh[z]$, and (iii) the rectangular function rect$[z]$, where:

$$\text{heaviside}[z] = \begin{cases} 0 & z < 0 \\ 1 & z \geq 0 \end{cases} \qquad \text{rect}[z] = \begin{cases} 0 & z < 0 \\ 1 & 0 \leq z \leq 1 \\ 0 & z > 1 \end{cases}. \tag{3.15}$$

Redraw a version of figure 3.3 for each of these functions. The original parameters were: $\boldsymbol{\phi} = \{\phi_0, \phi_1, \phi_2, \phi_3, \theta_{10}, \theta_{11}, \theta_{20}, \theta_{21}, \theta_{30}, \theta_{31}\} = \{-0.23, -1.3, 1.3, 0.66, -0.2, 0.4, -0.9, 0.9, 1.1, -0.7\}$. Provide an informal description of the family of functions that can be created by neural networks with one input, three hidden units, and one output for each activation function.

Problem 3.9* Show that the third linear region in figure 3.3 has a slope that is the sum of the slopes of the first and fourth linear regions.

Problem 3.10 Consider a neural network with one input, one output, and three hidden units. The construction in figure 3.3 shows how this creates four linear regions. Under what circumstances could this network produce a function with fewer than four linear regions?

Problem 3.11* How many parameters does the model in figure 3.6 have?

Problem 3.12 How many parameters does the model in figure 3.7 have?

Problem 3.13 What is the activation pattern for each of the seven regions in figure 3.8? In other words, which hidden units are active (pass the input) and which are inactive (clip the input) for each region?

Problem 3.14 Write out the equations that define the network in figure 3.11. There should be three equations to compute the three hidden units from the inputs and two equations to compute the outputs from the hidden units.

Problem 3.15* What is the maximum possible number of 3D linear regions that can be created by the network in figure 3.11?

Problem 3.16 Write out the equations for a network with two inputs, four hidden units, and three outputs. Draw this model in the style of figure 3.11.

Problem 3.17* Equations 3.11 and 3.12 define a general neural network with D_i inputs, one hidden layer containing D hidden units, and D_o outputs. Find an expression for the number of parameters in the model in terms of D_i, D, and D_o.

Problem 3.18* Show that the maximum number of regions created by a shallow network with $D_i = 2$-dimensional input, $D_o = 1$-dimensional output, and $D = 3$ hidden units is seven, as in figure 3.8j. Use the result of Zaslavsky (1975) that the maximum number of regions created by partitioning a D_i-dimensional space with D hyperplanes is $\sum_{j=0}^{D_i} \binom{D}{j}$. What is the maximum number of regions if we add two more hidden units to this model, so $D = 5$?