

Deep Neural Networks

Week 4 · Sessions 7–8 · September 15 & 17, 2026

Harun Pirim

Table of contents

1	Learning Objectives	2
2	Why Go Deep	3
3	Composing Two Networks	3
3.1	Folding the input space	7
4	From Composition to a Deep Network	9
5	Deep Neural Networks	10
5.1	Hyperparameters	13
6	Matrix Notation	13
6.1	The general formulation	14
7	Shallow vs. Deep Networks	16
7.1	Approximation power	16
7.2	Linear regions per parameter	16
7.3	Depth efficiency	18
7.4	Large, structured inputs	19
7.5	Training and generalization	19
8	Deep Networks in PyTorch	19
9	Summary	20
10	Exercises	21
11	Textbook Reference Map	23
12	Further Reading	24
Reading Prince, <i>Understanding Deep Learning</i> , Chapter 4 (pp. 41–55) · UDL notebooks 4.1–4.3		

! Assignment A2 — Shallow and Deep Networks

A formative notebook spanning Weeks 3 and 4: joints, activation patterns, and slopes of a shallow network; counting regions and parameters; composition, folding, and clipping; the matrix form of a deep network; and what a trained network actually realizes. Every task ends in a self-check that tells you at once whether the idea landed.

- [Open A2 in Google Colab](#) — nothing to install.

In Colab choose **File** → **Save a copy in Drive** before you start. When you are done, **Runtime** → **Run all**, then **File** → **Download** → **Download .ipynb** and upload that file on Blackboard under **Assignment A2**, named `A2_<lastname>.ipynb`. The deadline is posted on Blackboard.

i Review Quiz, Chapters 1–4 — Solution

The answer key to the in-class multiple-choice review of Weeks 1–4, with the rationale for each answer and the section and page in Prince where the idea is covered.

- [Download the solution \(PDF\)](#)

i How to read the references in these notes

Every section below opens with the section number and page range it covers in Prince, *Understanding Deep Learning* (MIT Press, 2024). Page numbers are those of the printed book. Figures taken from the textbook are captioned with their original figure number and page; figures generated by the code behind this page are ours. A complete cross-reference table appears in Section 11.

1 Learning Objectives

By the end of this week you should be able to:

1. Compose two shallow networks, predict how many linear regions the composition has, and explain the result in terms of **folding** the input space.
2. Show that a composition of two shallow networks is a special case of a network with two hidden layers, and say exactly what the general two-layer network can do that the composition cannot.
3. Describe the computation of a two-layer network as **clipping**: pre-activations with shared joints, ReLUs that add new joints, and a final linear recombination.
4. Distinguish parameters from **hyperparameters**, and define width, depth, and capacity.
5. Write a deep network of any depth in the matrix notation $\mathbf{h}_{k+1} = a[\boldsymbol{\beta}_k + \boldsymbol{\Omega}_k \mathbf{h}_k]$, and give the size of every weight matrix and bias vector.
6. Compare shallow and deep networks on approximation power, linear regions per parameter, depth efficiency, structured inputs, and training, and say which of these claims are proved and which are empirical.

7. Build a network of configurable depth in PyTorch, count its realized linear regions, and compare that count with the theoretical maximum.

💡 The code lives in a notebook

This page shows the figures and results; the Python behind them is not printed here. All of it — every cell, in the order the note uses it — is in a notebook you can open in the browser with nothing to install:

- [Open the Week 4 notebook in Google Colab](#) — all of this page’s code, in order.
- [UDL notebook 4.1 — Composing networks](#) — feed one shallow network into another and watch the regions multiply.
- [UDL notebook 4.2 — Clipping functions](#) — the second hidden layer, one panel of figure 4.5 at a time.
- [UDL notebook 4.3 — Deep networks](#) — the general matrix form of eq. 14 with arbitrary depth.

Colab gives you a Python environment with NumPy, matplotlib, and PyTorch already installed. In Colab, run a cell with **Shift+Enter**. If you prefer to work locally, `pip install -r requirements.txt` from the [course repository](#) gives you the same environment.

2 Why Go Deep

Prince, Chapter 4 opening, p. 41.

Week 3 ended on a strong result and a weak one. The strong result was the universal approximation theorem: a shallow network with enough hidden units can describe any continuous function on a compact set to any precision. The weak result was everything the theorem leaves out. It says nothing about *how many* hidden units “enough” is, and for some functions the answer is impractically large.

This chapter introduces **deep** networks, which have more than one hidden layer. With ReLU activations they still compute continuous piecewise linear functions, exactly as shallow networks do, so nothing about the *kind* of function changes. What changes is the exchange rate. Prince’s claim for the chapter is that “deep networks can produce many more linear regions than shallow networks for a given number of parameters” (p. 41), and hence describe a broader family of functions in practice, if not in principle. The rest of the week makes that claim precise, then asks whether it is actually the reason deep networks work.

3 Composing Two Networks

Prince §4.1, pp. 41–43; figures on pp. 42 and 44.

The cleanest way into deep networks is to take two shallow networks and feed the output of the first into the second (fig. 1a). Each has three hidden units. The first takes x and returns y :

$$h_1 = a[\theta_{10} + \theta_{11}x], \quad h_2 = a[\theta_{20} + \theta_{21}x], \quad h_3 = a[\theta_{30} + \theta_{31}x], \quad (1)$$

$$y = \phi_0 + \phi_1 h_1 + \phi_2 h_2 + \phi_3 h_3. \quad (2)$$

The second takes y and returns y' , with its own parameters, all primed:

$$h'_1 = a[\theta'_{10} + \theta'_{11}y], \quad h'_2 = a[\theta'_{20} + \theta'_{21}y], \quad h'_3 = a[\theta'_{30} + \theta'_{31}y], \quad (3)$$

$$y' = \phi'_0 + \phi'_1 h'_1 + \phi'_2 h'_2 + \phi'_3 h'_3. \quad (4)$$

With ReLU activations the composition is still a piecewise linear function of x . The question is how many pieces. A shallow network with six hidden units has at most seven regions. The composition, which also has six hidden units, can have more.

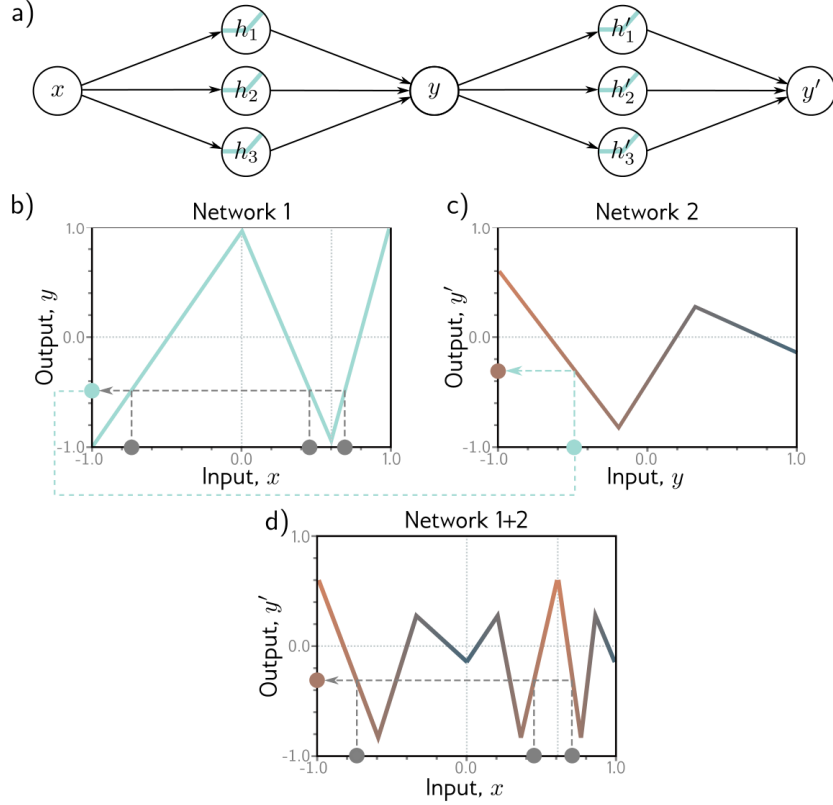


Figure 4.1 Composing two single-layer networks with three hidden units each. a) The output y of the first network constitutes the input to the second network. b) The first network maps inputs $x \in [-1, 1]$ to outputs $y \in [-1, 1]$ using a function comprising three linear regions that are chosen so that they alternate the sign of their slope. 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).

Figure 1: Prince (2024), figure 4.1, p. 42.

The mechanism is in fig. 1b–d. Choose the first network so that its three regions alternate in slope and each one sweeps the whole output range $y \in [-1, 1]$. Then three different input ranges are mapped onto the *same* output range, and whatever the second network does to $y \in [-1, 1]$ happens three times over as x runs from -1 to 1 , “variously flipped and rescaled according to the slope of the regions” (p. 42). The second network’s three regions are copied three times: nine linear regions from six hidden units.

Our version follows, with parameters chosen so that the counts can be checked. The first network is a zigzag with slopes $3, -3, 3$ and joints at $\pm 1/3$; the second is any three-region function on $[-1, 1]$. The region count is done by the method of Week 3: sample the input densely and count distinct activation patterns across *all six* hidden units, since every linear region of the composition has its own pattern.

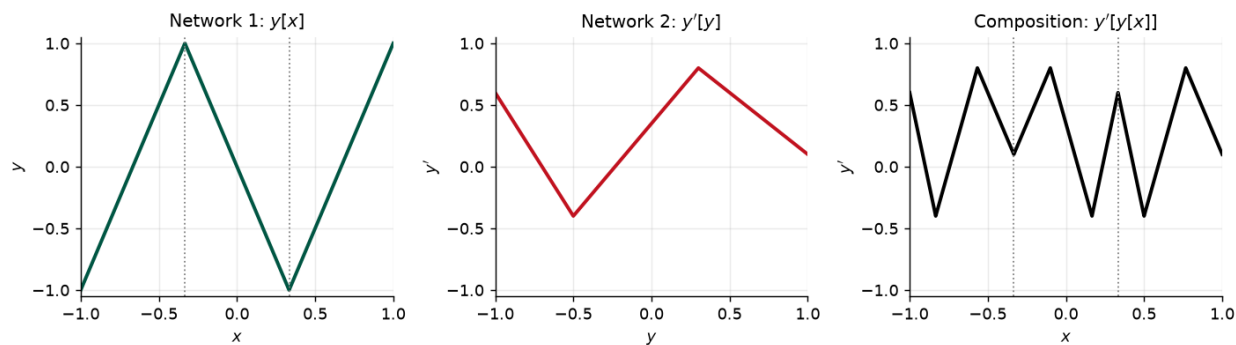


Figure 2: Our version of figure 4.1b–d. Left: the first network maps $x \in [-1, 1]$ onto $y \in [-1, 1]$ three times, once per region. Middle: the second network’s three-region function of y . Right: the composition. The middle panel’s shape appears three times across the input, once per region of the first network, alternately reversed; the printed count confirms nine regions from six hidden units.

```

network 1 alone : 3 regions from 3 hidden units
network 2 alone : 3 regions from 3 hidden units
composition      : 9 regions from 6 hidden units
shallow, 6 units: at most 7

```

The same principle applies in higher dimensions (fig. 3). There the first network is the two-input, three-unit network of Week 3, whose output surface has seven regions, one of them flat. Passing that through a second network with two regions splits each of the six non-flat regions in two, for thirteen in all. The flat region is not split, because a constant input gives the second network nothing to bend.

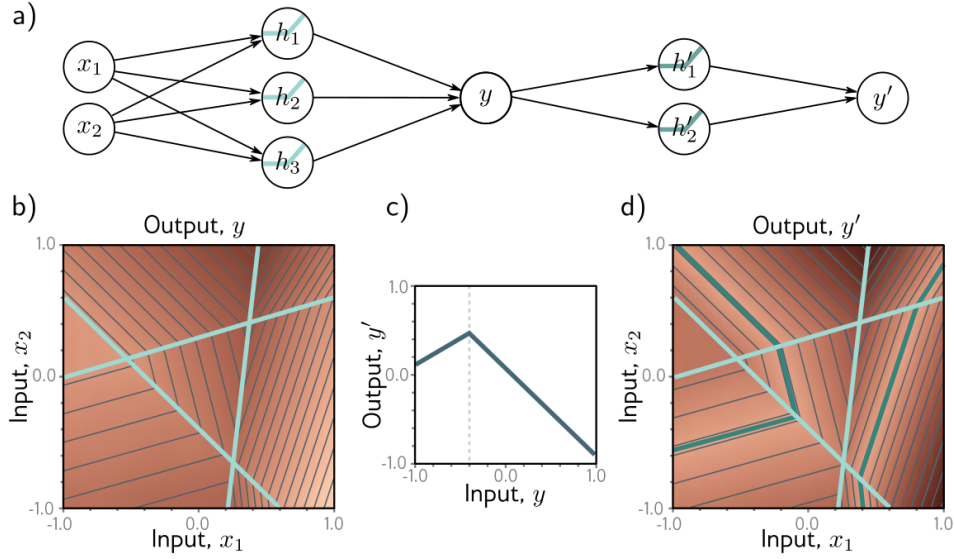


Figure 4.2 Composing neural networks with a 2D input. a) The first network (from figure 3.8) has three hidden units and takes two inputs x_1 and x_2 and returns a scalar output y . This is passed into a second network with two hidden units to produce y' . b) The first network produces a function consisting 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 3: Prince (2024), figure 4.2, p. 44.

3.1 Folding the input space

Prince §4.1, p. 43; figure on p. 44.

Prince offers a second picture of the same construction (fig. 4). The first network **folds** the input space back on itself, so that several inputs land on the same output. The second network then draws its function once on the folded paper. Unfolding reveals that function replicated at every point that had been stacked together, mirrored at each crease. The picture is worth keeping in mind for two reasons. It explains where the multiplication of regions comes from, and it exposes the price: the copies are not independent. The composition has nine regions but only twenty parameters, and the shapes of the nine pieces are tied to one another by the fold.

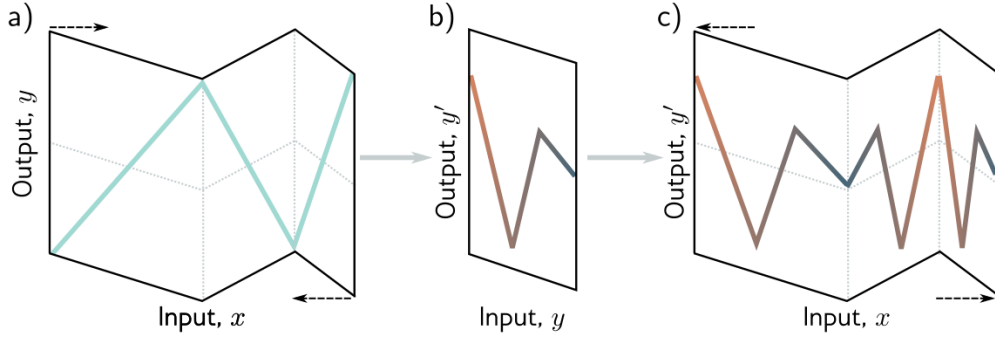


Figure 4.3 Deep networks as folding input space. a) One way to think about the first network from figure 4.1 is that it “folds” the input space back on top of itself. b) The second network applies its function to the folded space. c) The final output is revealed by “unfolding” again.

Figure 4: Prince (2024), figure 4.3, p. 44.

If one fold triples the regions, more folds should multiply them again. Fig. 5 composes the zigzag network with *itself* K times. Each composition adds three hidden units and, in the matrix notation of §4.4 below, twelve parameters. The regions go up by a factor of three each time.

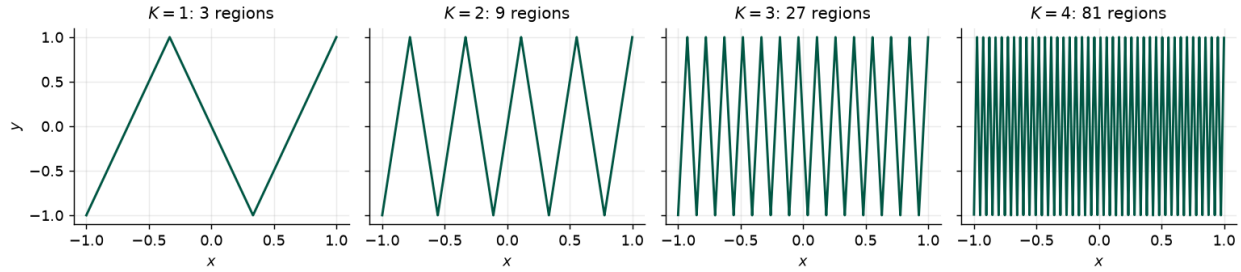


Figure 5: The zigzag network of fig. 2 composed with itself one to four times. Every layer folds the input three ways, so the region count triples with each layer while the hidden-unit count grows by only three. The printed table compares the count of activation patterns with 3^K and with the most a shallow network of the same size could manage.

layers	K	hidden units	parameters	regions	3^K	shallow max
	1	3	10	3	3	4
	2	6	22	9	9	7
	3	9	34	27	27	10
	4	12	46	81	81	13

Twelve hidden units arranged as four layers of three make eighty-one regions. Twelve hidden units in a single layer make at most thirteen. That factor is the whole case for depth in one table, and the caveat that goes with it is in the figure: the eighty-one pieces are all the same triangle, at four scales. We return to this in §4.5.

4 From Composition to a Deep Network

Prince §4.2, pp. 43–45; figure on p. 45.

Composing two networks turns out to be a special case of something simpler. The output of the first network, eq. 2, is linear in the hidden units h_1, h_2, h_3 . The first thing the second network does to y , inside the brackets of eq. 3, is also linear. A linear function of a linear function is linear, so substituting eq. 2 into eq. 3 gives

$$\begin{aligned} h'_1 &= a[\theta'_{10} + \theta'_{11}y] = a[\theta'_{10} + \theta'_{11}\phi_0 + \theta'_{11}\phi_1h_1 + \theta'_{11}\phi_2h_2 + \theta'_{11}\phi_3h_3] \\ h'_2 &= a[\theta'_{20} + \theta'_{21}y] = a[\theta'_{20} + \theta'_{21}\phi_0 + \theta'_{21}\phi_1h_1 + \theta'_{21}\phi_2h_2 + \theta'_{21}\phi_3h_3] \\ h'_3 &= a[\theta'_{30} + \theta'_{31}y] = a[\theta'_{30} + \theta'_{31}\phi_0 + \theta'_{31}\phi_1h_1 + \theta'_{31}\phi_2h_2 + \theta'_{31}\phi_3h_3], \end{aligned} \quad (5)$$

which, after renaming the constants, is

$$\begin{aligned} h'_1 &= a[\psi_{10} + \psi_{11}h_1 + \psi_{12}h_2 + \psi_{13}h_3] \\ h'_2 &= a[\psi_{20} + \psi_{21}h_1 + \psi_{22}h_2 + \psi_{23}h_3] \\ h'_3 &= a[\psi_{30} + \psi_{31}h_1 + \psi_{32}h_2 + \psi_{33}h_3], \end{aligned} \quad (6)$$

with $\psi_{10} = \theta'_{10} + \theta'_{11}\phi_0$, $\psi_{11} = \theta'_{11}\phi_1$, $\psi_{12} = \theta'_{11}\phi_2$, and so on. The intermediate variable y has disappeared. What is left is a network with **two hidden layers**, each of three units, in which every unit of the first layer feeds every unit of the second (fig. 6).

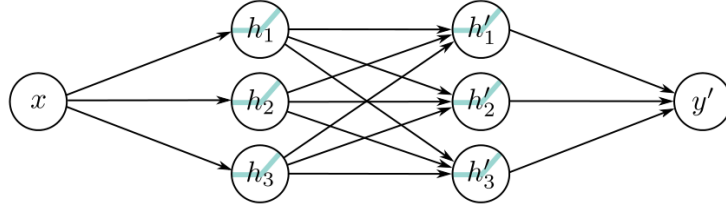


Figure 4.4 Neural network with one input, one output, and two hidden layers, each containing three hidden units.

Figure 6: Prince (2024), figure 4.4, p. 45.

So a two-layer network can represent anything the composition can. It can also represent more. In eq. 6 the nine slopes $\psi_{11}, \dots, \psi_{33}$ are free, whereas in eq. 5 they are forced to be the **outer product**

$$\Psi = \begin{bmatrix} \theta'_{11} \\ \theta'_{21} \\ \theta'_{31} \end{bmatrix} [\phi_1 \quad \phi_2 \quad \phi_3], \quad (7)$$

a 3×3 matrix of rank one. The composition squeezes the three hidden units of the first layer through a single number y before the second layer sees them; the general two-layer network does

not. We can check both halves of this claim, that the composition equals the two-layer network with the outer-product Ψ , and that this Ψ is rank-deficient.

```
max |composition - two-layer network| = 2.2e-16
rank of the outer-product Psi          = 1    (a free 3x3 Psi has rank up to 3)
```

5 Deep Neural Networks

Prince §4.3, pp. 45–47; figure on p. 47.

Now take the general two-layer network of fig. 6 at face value. The first layer is

$$h_1 = a[\theta_{10} + \theta_{11}x], \quad h_2 = a[\theta_{20} + \theta_{21}x], \quad h_3 = a[\theta_{30} + \theta_{31}x], \quad (8)$$

the second layer is eq. 6, which we restate as

$$\begin{aligned} h'_1 &= a[\psi_{10} + \psi_{11}h_1 + \psi_{12}h_2 + \psi_{13}h_3] \\ h'_2 &= a[\psi_{20} + \psi_{21}h_1 + \psi_{22}h_2 + \psi_{23}h_3] \\ h'_3 &= a[\psi_{30} + \psi_{31}h_1 + \psi_{32}h_2 + \psi_{33}h_3], \end{aligned} \quad (9)$$

and the output is

$$y' = \phi'_0 + \phi'_1 h'_1 + \phi'_2 h'_2 + \phi'_3 h'_3. \quad (10)$$

Reading these three equations in order gives a second picture of how depth builds complexity, this time in terms of **clipping** rather than folding (fig. 7):

1. The first-layer hidden units h_1, h_2, h_3 are computed as in Week 3: three lines, each clipped by a ReLU (eq. 8).
2. The **pre-activations** of the second layer are three new linear functions of those hidden units, the arguments of the ReLUs in eq. 9. At this point we have, in effect, a shallow network with three outputs, and Week 3 told us what that looks like: three piecewise linear functions whose joints are all in the same places (fig. 7a–c).
3. The second-layer ReLU clips each of these functions wherever it dips below zero. Every crossing is a **new joint** that the first layer did not have (fig. 7d–f).
4. The output is a linear combination of the clipped functions, eq. 10 (fig. 7g–j).

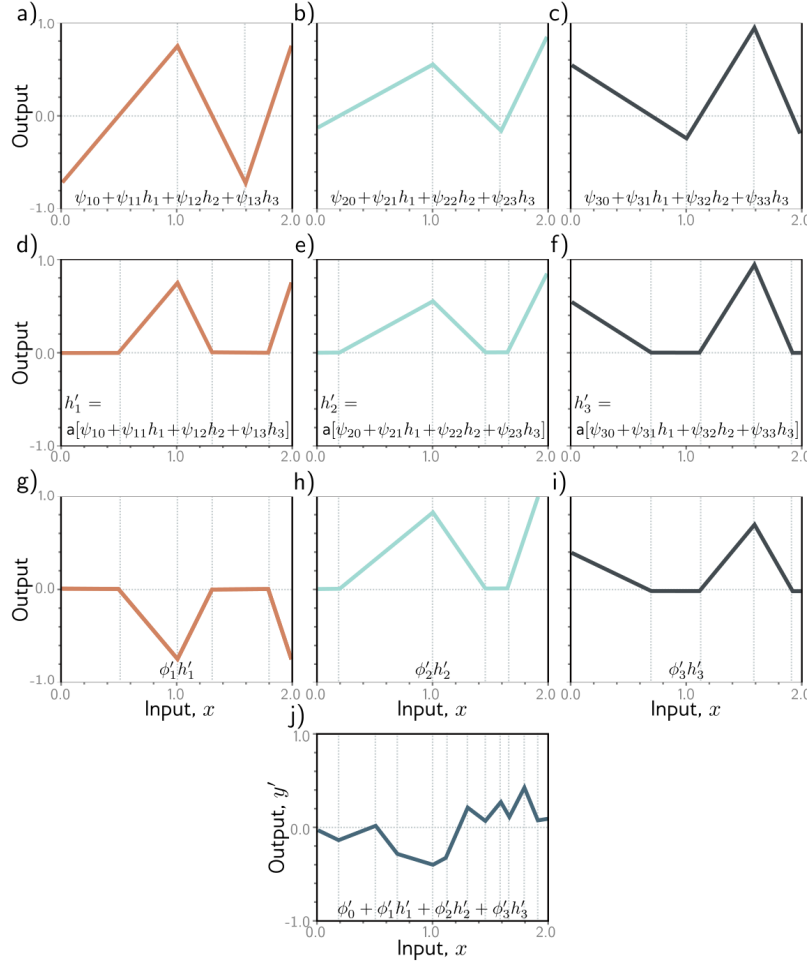


Figure 4.5 Computation for the deep network in figure 4.4. a–c) The inputs to the second hidden layer (i.e., the pre-activations) are three piecewise linear functions where the “joints” between the linear regions are at the same places (see figure 3.6). d–f) Each piecewise linear function is clipped to zero by the ReLU activation function. g–i) These clipped functions are then weighted with parameters $\phi'_1, \phi'_2,$ 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 7: Prince (2024), figure 4.5, p. 47.

Fig. 8 is our version, with the first layer taken from Prince’s own figure 3.3 parameters (problem 3.8), so its three joints sit at $x = 0.5, 1,$ and 1.57 as they did last week. The second layer’s parameters are chosen so that each pre-activation crosses zero twice inside the plot. The code then counts the joints layer by layer.

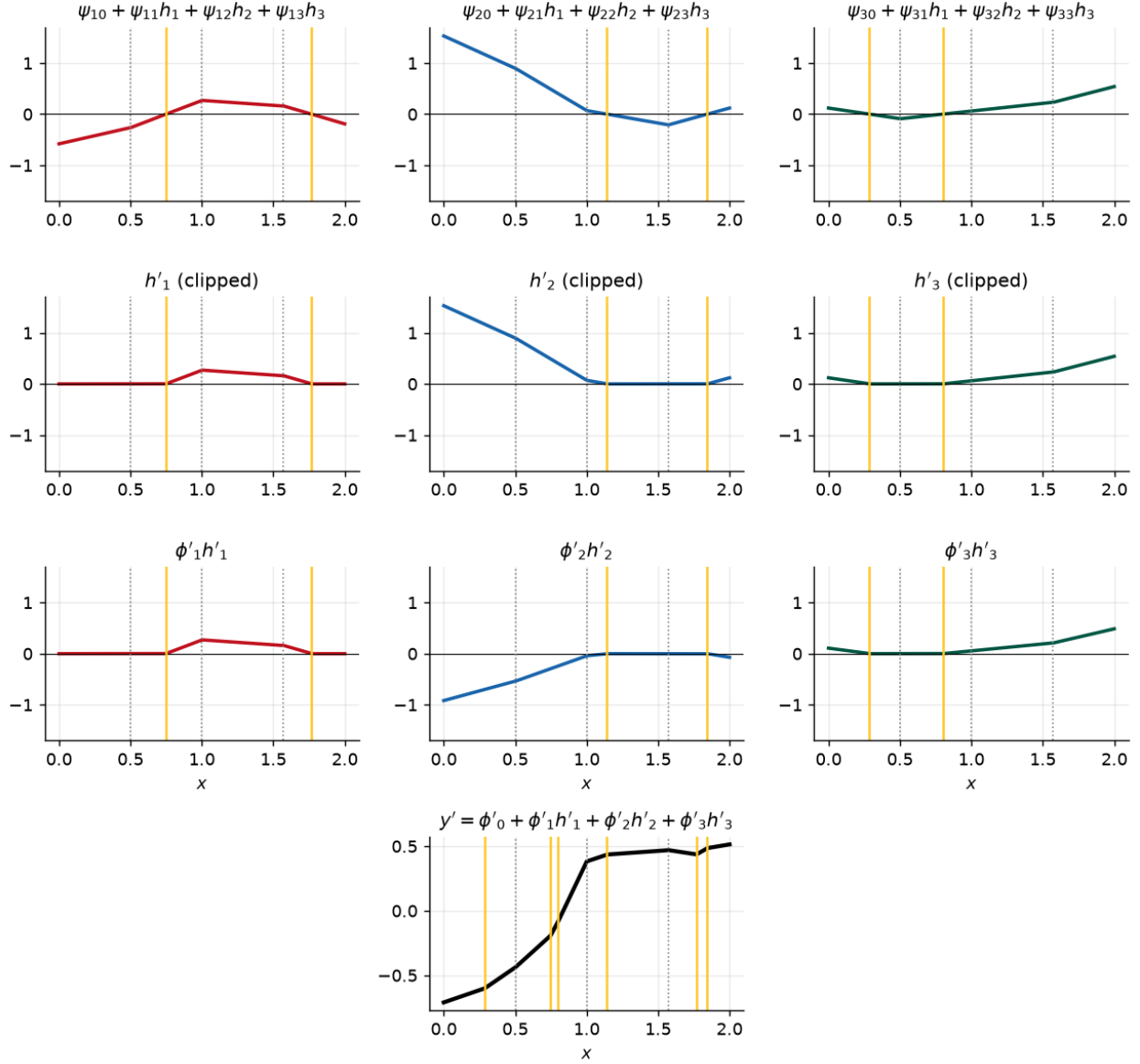


Figure 8: Our version of figure 4.5. Top row: the three second-layer pre-activations, piecewise linear functions of x that share the first layer's three joints (dotted grey). Second row: each is clipped by a ReLU, which adds a joint (gold) wherever it crosses zero. Third row: the clipped functions weighted by $\phi'_1, \phi'_2, \phi'_3$. Bottom: their sum plus ϕ'_0 . The output has every joint from both rows.

joints from layer 1	: 3
new joints added by layer 2	: 6 (2, 2, 2 per unit)
linear regions in the output	: 10 (= 1 + 3 + 6)
shallow network with 6 units	: at most 7

The joint count is the point. Six hidden units in one layer give at most six joints, and every joint is a zero-crossing of a *line*, so there is exactly one per unit. Six hidden units in two layers give three

joints from the first layer plus as many from the second as its pre-activations can be made to cross zero, and those pre-activations are already piecewise linear, so each can cross several times. New joints multiply rather than add.

! Two pictures, neither complete

Folding and clipping describe the same computation. Folding shows the *dependencies* among the regions — the copies made by a fold are tied together — but hides how the new joints arise. Clipping shows the new joints but hides the dependencies. Prince is explicit that “both descriptions provide only partial insight into how deep neural networks operate” (p. 46), and that what we actually have is still just an equation from x to y' . Written out in full by substituting eq. 8 into eq. 9 into eq. 10, it is

$$\begin{aligned} y' = & \phi'_0 + \phi'_1 a[\psi_{10} + \psi_{11}a[\theta_{10} + \theta_{11}x] + \psi_{12}a[\theta_{20} + \theta_{21}x] + \psi_{13}a[\theta_{30} + \theta_{31}x]] \\ & + \phi'_2 a[\psi_{20} + \psi_{21}a[\theta_{10} + \theta_{11}x] + \psi_{22}a[\theta_{20} + \theta_{21}x] + \psi_{23}a[\theta_{30} + \theta_{31}x]] \\ & + \phi'_3 a[\psi_{30} + \psi_{31}a[\theta_{10} + \theta_{11}x] + \psi_{32}a[\theta_{20} + \theta_{21}x] + \psi_{33}a[\theta_{30} + \theta_{31}x]], \end{aligned} \quad (11)$$

“although this is admittedly rather difficult to understand” (p. 46). The matrix notation of the next section exists so that nobody has to write it.

5.1 Hyperparameters

Prince §4.3.1, p. 46.

Nothing stops us adding a third hidden layer, or a hundredth; modern networks may have more than a hundred layers with thousands of units in each. Three words fix the vocabulary. The number of hidden units in a layer is the network’s **width**, the number of hidden layers its **depth**, and the total number of hidden units is a measure of its **capacity**.

We write K for the number of hidden layers and $D_1, D_2, \dots, D_K$ for the number of units in each. These are **hyperparameters**: quantities chosen *before* the parameters (the slopes and intercepts) are learned. The distinction matters for how we think about a network. For fixed hyperparameters, say $K = 2$ and $D_1 = D_2 = 3$, the model is a family of functions and the parameters select one member. Letting the hyperparameters vary as well, a neural network is “a family of families of functions relating input to output” (p. 46). Chapter 8 takes up how to choose them.

6 Matrix Notation

Prince §4.4, pp. 48–49; figure on p. 48.

A deep network is linear transformations alternating with activation functions, and that structure is what matrix notation was built for (Appendix B.3; Week 2). Eq. 8, eq. 9, and eq. 10 become

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

and $y' = \phi'_0 + [\phi'_1 \ \phi'_2 \ \phi'_3] [h'_1 \ h'_2 \ h'_3]^T$, or more compactly

$$\mathbf{h} = a[\boldsymbol{\theta}_0 + \boldsymbol{\theta}x], \quad \mathbf{h}' = a[\boldsymbol{\psi}_0 + \boldsymbol{\Psi}\mathbf{h}], \quad y' = \phi'_0 + \boldsymbol{\phi}'\mathbf{h}', \quad (13)$$

where $a[\cdot]$ applies the activation function separately to every element of its vector argument.

6.1 The general formulation

Prince §4.4.1, pp. 48–49.

Primes and a fresh Greek letter per layer do not scale to a hundred layers. From here on the book, and this course, use one naming scheme for every layer. The vector of hidden units at layer k is $\mathbf{h}_k$. The biases that feed layer $k + 1$ are $\boldsymbol{\beta}_k$, and the weights that act on layer k to produce the pre-activations of layer $k + 1$ are $\boldsymbol{\Omega}_k$. A deep network $\mathbf{y} = \mathbf{f}[\mathbf{x}, \boldsymbol{\phi}]$ with K hidden layers is then

$$\begin{aligned} \mathbf{h}_1 &= a[\boldsymbol{\beta}_0 + \boldsymbol{\Omega}_0 \mathbf{x}] \\ \mathbf{h}_2 &= a[\boldsymbol{\beta}_1 + \boldsymbol{\Omega}_1 \mathbf{h}_1] \\ \mathbf{h}_3 &= a[\boldsymbol{\beta}_2 + \boldsymbol{\Omega}_2 \mathbf{h}_2] \\ &\vdots \\ \mathbf{h}_K &= a[\boldsymbol{\beta}_{K-1} + \boldsymbol{\Omega}_{K-1} \mathbf{h}_{K-1}] \\ \mathbf{y} &= \boldsymbol{\beta}_K + \boldsymbol{\Omega}_K \mathbf{h}_K, \end{aligned} \quad (14)$$

and the parameters are every one of these matrices and vectors, $\boldsymbol{\phi} = \{\boldsymbol{\beta}_k, \boldsymbol{\Omega}_k\}_{k=0}^K$. Note that the output line has no activation function: the last step is purely linear, as it was for the shallow network.

i Notation

The index on $\boldsymbol{\beta}_k$ and $\boldsymbol{\Omega}_k$ is the layer they are applied *to*, not the layer they produce: $\boldsymbol{\Omega}_0$ acts on the input $\mathbf{x}$ (layer zero) and $\boldsymbol{\Omega}_K$ acts on the last hidden layer to produce $\mathbf{y}$. A network with K hidden layers therefore has $K + 1$ weight matrices and $K + 1$ bias vectors.

Every size follows from where each object sits (fig. 9). If layer k has D_k hidden units, then the bias vector $\boldsymbol{\beta}_{k-1}$ that feeds it has D_k entries, and the last bias vector $\boldsymbol{\beta}_K$ has D_o entries, one per output. The first weight matrix $\boldsymbol{\Omega}_0$ is $D_1 \times D_i$, the last, $\boldsymbol{\Omega}_K$, is $D_o \times D_K$, and each one in between, $\boldsymbol{\Omega}_k$, is $D_{k+1} \times D_k$: rows for the layer it produces, columns for the layer it consumes.

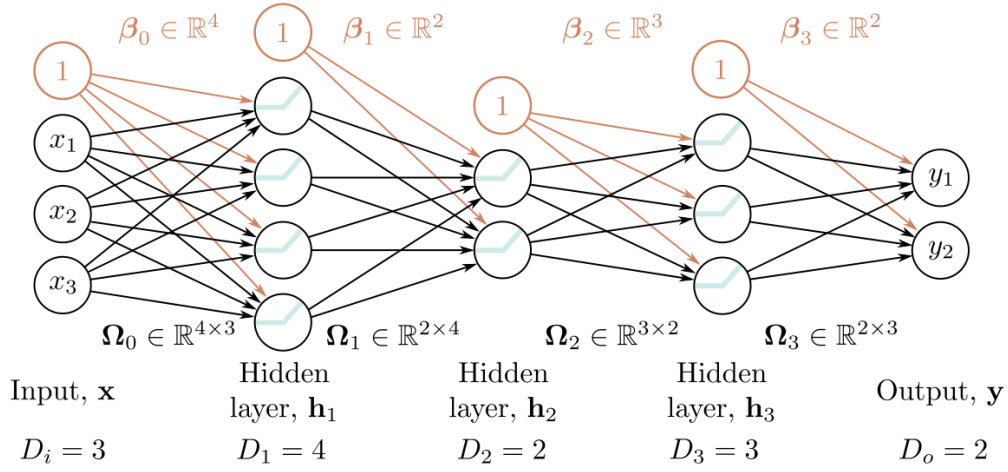


Figure 4.6 Matrix notation for network with $D_i = 3$ -dimensional input $\mathbf{x}$, $D_o = 2$ -dimensional output $\mathbf{y}$, and $K = 3$ hidden layers $\mathbf{h}_1, \mathbf{h}_2$, and $\mathbf{h}_3$ of dimensions $D_1 = 4$, $D_2 = 2$, and $D_3 = 3$ respectively. The weights are stored in matrices Ω_k that pre-multiply the activations from the preceding layer to create the pre-activations at the subsequent layer. For example, the weight matrix Ω_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 β_k and have the dimension of the layer into which they feed. For example, the bias vector β_2 is length three because layer $\mathbf{h}_3$ contains three hidden units.

Figure 9: Prince (2024), figure 4.6, p. 48.

Finally, the whole network can be written as one nested expression,

$$\mathbf{y} = \beta_K + \Omega_K a \left[\beta_{K-1} + \Omega_{K-1} a \left[\dots \beta_2 + \Omega_2 a \left[\beta_1 + \Omega_1 a \left[\beta_0 + \Omega_0 \mathbf{x} \right] \dots \right] \right] \right], \quad (15)$$

which is eq. 11 with the subscripts tidied up, and no easier to read. In PyTorch, though, it is one line. A `Linear` layer is $\beta_k + \Omega_k \mathbf{h}_k$, and its `weight` and `bias` attributes are the matrix and vector exactly, with the shapes just described. The following builds the network of fig. 9 and prints every shape, which is the check problem 4.2 asks for.

```

layer      Omega_k shape  beta_k size  reads
k = 0      (4, 3)      4           h_1 = a[beta_0 + Omega_0 x]
k = 1      (2, 4)      2           h_2 = a[beta_1 + Omega_1 h_1]
k = 2      (3, 2)      3           h_3 = a[beta_2 + Omega_2 h_2]
k = 3      (2, 3)      2           y = beta_3 + Omega_3 h_3

```

total parameters: 43

Row counts match the layer each matrix produces and column counts the layer it consumes, as the figure says: Ω_1 is 2×4 because it turns four units into two. Summing entries gives the parameter count, which for the general case with D_i inputs, D_o outputs, and widths $D_1, \dots, D_K$ is

$$\underbrace{D_1(D_1 + 1)}_{\Omega_0, \beta_0} + \sum_{k=1}^{K-1} \underbrace{D_{k+1}(D_k + 1)}_{\Omega_k, \beta_k} + \underbrace{D_o(D_K + 1)}_{\Omega_K, \beta_K}. \quad (16)$$

7 Shallow vs. Deep Networks

Prince §4.5, pp. 49–51; figure on p. 51.

Week 3 described shallow networks and this week deep ones. Prince compares them along five lines, and it is worth being clear about which comparisons are theorems, which are counting arguments, and which are observations.

7.1 Approximation power

Prince §4.5.1, p. 50.

Both can approximate any continuous function. The shallow case is the universal approximation theorem of Week 3. The deep case follows from it: a two-layer network contains the composition of two shallow networks as a special case, and if the second of those computes the identity function (problem 4.7), the deep network *is* a shallow network. So a deep network with enough capacity can approximate any continuous function arbitrarily closely. This is a theorem, and by itself it says nothing about which architecture is better.

7.2 Linear regions per parameter

Prince §4.5.2, p. 50; notes on pp. 52–53.

Here the two differ. For one input and one output, a shallow network with $D > 2$ hidden units makes at most $D + 1$ linear regions from $3D + 1$ parameters. A deep network with K layers of $D > 2$ units each makes up to

$$(D + 1)^K \text{ regions from } 3D + 1 + (K - 1)D(D + 1) \text{ parameters.} \quad (17)$$

Regions grow exponentially in depth; parameters grow only linearly. The parameter count is eq. 16 with $D_i = D_o = 1$ and every width equal to D (problem 4.10), and the region count is the folding argument of §4.1 done K times: each layer can fold the input into $D + 1$ pieces on which the next layer's function is replicated, just as fig. 5 folded it into three. Fig. 10a plots regions against parameters for depths one to five, and the gap widens with every layer. The book's headline example is a network with $K = 5$ and $D = 10$: 471 parameters and 161,051 regions.

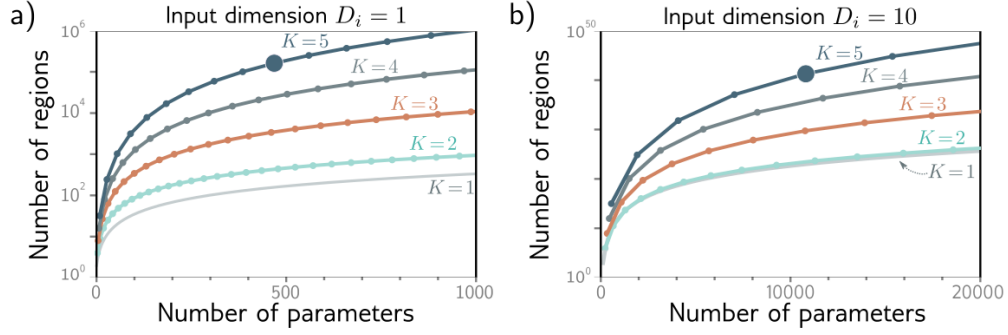


Figure 4.7 The maximum number of linear regions for neural networks increases rapidly with the network depth. a) Network with $D_i = 1$ input. Each curve represents a fixed number of hidden layers K , as we vary the number of hidden units D per layer. For a fixed parameter budget (horizontal position), 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.

Figure 10: Prince (2024), figure 4.7, p. 51.

With more inputs the maximum is harder to compute, but the chapter notes (p. 53) give a closed form for the case where every layer has the same width D and D is a multiple of D_i :

$$N_r = \left(\frac{D}{D_i} + 1 \right)^{D_i(K-1)} \sum_{j=0}^{D_i} \binom{D}{j}. \quad (18)$$

The first factor is the first $K - 1$ layers folding the space, with D/D_i units now needed per input dimension to make each fold. The sum is the Zaslavsky bound of Week 3, the regions the final layer carves out of each folded piece. Fig. 11 evaluates both formulas and reproduces the two points the book highlights.

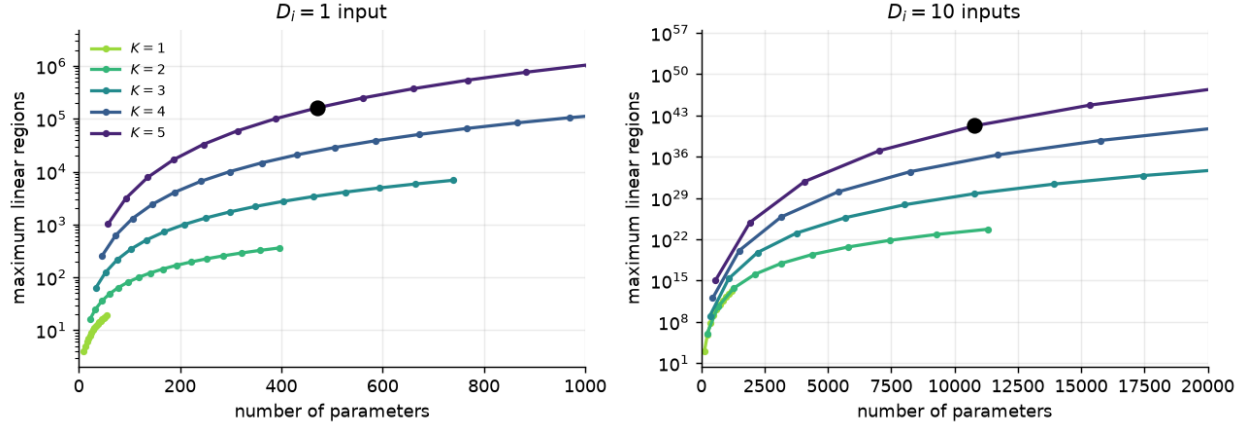


Figure 11: Our version of figure 4.7. Left: one input, maximum regions $(D+1)^K$ against parameters $3D+1+(K-1)D(D+1)$ for $K=1, \dots, 5$ hidden layers, sweeping the width D . Right: ten inputs, using eq. 18, in steps of ten units. At any fixed parameter budget the deeper curve is higher, and the vertical axis is logarithmic. Black dots are the two networks the book singles out.

$K = 5, D = 10, D_i = 1$: 471 parameters, 161,051 regions
 $K = 5, D = 50, D_i = 10$: 10,801 parameters, about 10^{41} regions

! Regions are not free capacity

This is the place to be careful. A deep network’s regions “contain complex dependencies and symmetries” (p. 50): the folding picture showed that the copies are mirror images of one another, and fig. 5 was eighty-one regions made of one triangle. The flexibility of the function is still bounded by the number of parameters, and forty-six parameters cannot describe eighty-one independent pieces. Prince’s conclusion is conditional: the extra regions are an advantage only if (i) the function we want to approximate has similar symmetries, or (ii) we have reason to believe the mapping from input to output “really does involve a composition of simpler functions” (p. 51). For many engineering systems (ii) is a reasonable belief, and it is a better argument for depth than the raw count is.

7.3 Depth efficiency

Prince §4.5.3, p. 50; notes on p. 53.

There is a stronger theoretical statement than region counting. For certain functions, a shallow network needs **exponentially** more hidden units than a deep network to reach the same approximation error. This is the **depth efficiency** of neural networks, and the chapter notes trace it through Telgarsky (2016), who builds one-input functions realizable with $O[k^3]$ layers of constant width that no network of $O[k]$ layers and width below 2^k can match, and Eldan & Shamir (2016), who show that with multivariate inputs some three-layer networks cannot be realized by any two-layer network of sub-exponential size. The corresponding **width efficiency** results (Lu et al., 2017) are weaker: a wide shallow network needs at most a *polynomial* increase in depth to be matched by

a narrow one. Depth is the more valuable resource. The same caveat applies as before, though: it is not clear that the functions we meet in practice belong to the classes these theorems describe.

7.4 Large, structured inputs

Prince §4.5.4, pp. 50–51.

Everything so far has been fully connected: every unit in one layer feeds every unit in the next. For an image with a million pixels that is not practical, and the reason is not only the parameter count. We want different parts of the image to be processed the *same* way, because there is no point learning to recognize an object separately at every position it might appear. The solution is to process local regions in parallel and then integrate information from progressively larger regions. Such local-to-global processing is hard to specify without multiple layers, and it is the subject of Chapter 10.

```
fully connected first layer, 100 hidden units: 300,000,100 parameters ( 1.2 GB in float32)
fully connected first layer, 1000 hidden units: 3,000,001,000 parameters ( 12.0 GB in float32)
```

One layer of a thousand units, on one photograph, already needs twelve gigabytes of weights, and every one of those weights would have to be learned from data. The count is the least of the problems; the missing structure is the real one.

7.5 Training and generalization

Prince §4.5.5, p. 51.

The last two comparisons are empirical. Moderately deep networks are usually *easier* to fit than shallow ones, perhaps because over-parameterized deep models have a large family of roughly equivalent solutions that are easy to find. Past a certain depth, training becomes difficult again, and much of Chapter 11 is about the methods that mitigate this. Deep networks also seem to **generalize** better to new data. In practice the best results on most tasks come from networks with tens or hundreds of layers, and “neither of these phenomena are well understood” (p. 51); Chapter 20 returns to both.

8 Deep Networks in PyTorch

Ours; UDL notebook 4.3.

The theory of §4.5 is about what a network *can* represent. Whether gradient descent finds it is a different question, and Chapters 6 and 7 are about the answer. We can preview the gap now. Problem 4.11 poses two networks with the same parameter count: a shallow one with $D = 95$ hidden units and a deep one with $K = 10$ layers of $D = 5$. By eq. 17 the shallow network can make 96 regions and the deep one about sixty million. We fit both, with the Adam optimizer of Week 2, to noisy samples of a triangle wave, a target whose repeated symmetric pieces are exactly the kind of structure a fold reproduces for free. Then we count the regions each trained network *actually* uses, by tallying activation patterns across every hidden unit, as in fig. 2.

network	parameters	max regions	realized	final MSE
shallow, $D = 95$	286	96	33	0.0022
deep, $K = 10, D = 5$	286	60,466,176	22	0.3227

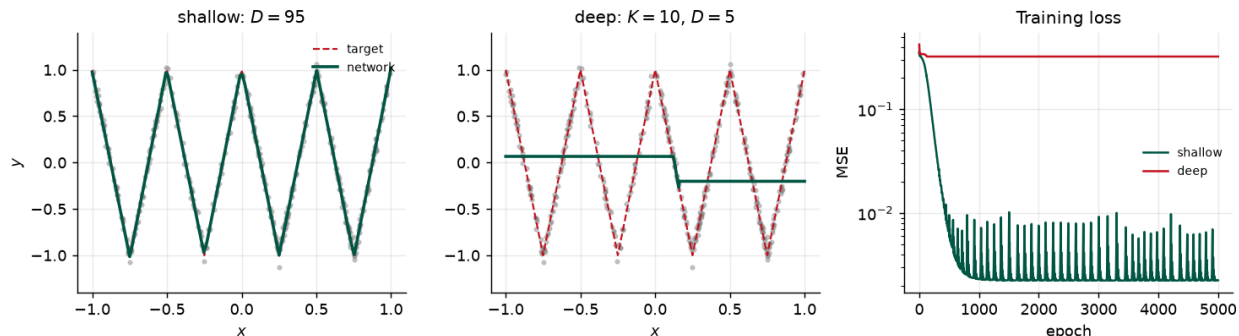


Figure 12: Two 286-parameter networks fitted to the same noisy triangle wave. Left: the shallow network follows the target closely. Middle: the deep, narrow network does not, despite a theoretical capacity hundreds of thousands of times larger. Right: training loss for both. Depth changes what a network can represent; it also changes how hard the parameters are to find, which is the subject of Chapters 6, 7, and 11.

Two things stand out. Neither network comes anywhere near its maximum region count: the shallow one realizes about half of its 96, and the deep one a couple of dozen of its sixty million. And on this target, with this optimizer and this much training, the shallow network wins outright. The maximum in eq. 17 is a statement about the *best* parameter setting; gradient descent from a random start does not find it, and for a narrow, deep ReLU network it barely gets started, for reasons Chapter 7 will make precise when we look at how gradients propagate through many layers. Meanwhile the hand-built network of fig. 5, with forty-six parameters, would fit a triangle wave exactly. The capacity is real. Reaching it is the hard part, and it is why the remainder of the course is mostly about training.

9 Summary

Prince §4.6, p. 52.

Composing two shallow networks makes the first network **fold** the input space and the second apply a piecewise linear function that is duplicated wherever the space was folded onto itself. That composition is a special case of a network with two hidden layers, in which the second-layer ReLUs **clip** the incoming functions and add joints. Hyperparameters — the depth K and the widths $D_1, \dots, D_K$ — fix the family; parameters $\{\beta_k, \Omega_k\}$ fix the member, and the general network is $K + 1$ affine maps with a ReLU between each pair, eq. 14.

Comparing shallow with deep: (i) both approximate any continuous function given enough capacity; (ii) deep networks produce many more linear regions per parameter, though the regions are tied together by the folds; (iii) some functions need exponentially more shallow units than deep ones; (iv) large, structured inputs like images are best handled in stages, which needs layers; and (v) in

practice the best results come from deep networks with many layers, for reasons that are not yet understood.

Next: we now know what shallow and deep networks can represent, and we have seen that representing a function and finding it are different problems. Chapter 5 introduces loss functions, which turn “how wrong is this model” into a single number for any choice of parameters. Chapters 6 and 7 then describe the search for the parameters that minimize it.

10 Exercises

Problem numbers refer to Prince, Chapter 4 (pp. 53–55). Starred UDL problems have published answers on the [UDL website](#). The computational problems extend code that is already written for you — start from the [Colab notebook](#).

1. (UDL 4.1*) Consider composing the two networks in fig. 13. Sketch the relationship between x and y' for $x \in [-1, 1]$. How many linear regions does it have, and how does that follow from the number of regions in panels (b) and (c)? Then check your sketch in the notebook by encoding both panels as shallow networks and composing them as fig. 2 does.

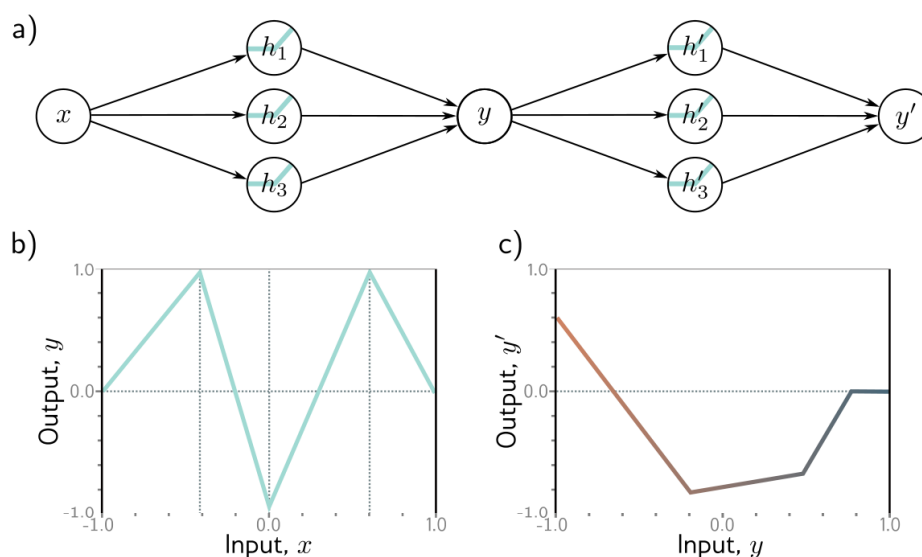


Figure 4.8 Composition of two networks for problem 4.1. a) The output y of the first network becomes the input to the second. b) The first network computes this function with output values $y \in [-1, 1]$. c) The second network computes this function on the input range $y \in [-1, 1]$.

Figure 13: Prince (2024), figure 4.8, p. 54.

2. (UDL 4.2, 4.5, and 4.6) Identify the four hyperparameters in fig. 9. For a network with $D_i = 5$ inputs, $D_o = 1$ output, and $K = 20$ hidden layers of $D = 30$ units each, state the depth

and the width. Then consider a network with $D_i = D_o = 1$ and $K = 10$ layers of $D = 10$ units: would the number of weights increase more if the depth grew by one or if the width grew by one? Show your reasoning using eq. 16.

3. (UDL 4.3) Use the non-negative homogeneity of the ReLU (problem 3.5) to show that for non-negative scalars λ_0, λ_1 ,

$$\text{ReLU}[\beta_1 + \lambda_1 \Omega_1 \text{ReLU}[\beta_0 + \lambda_0 \Omega_0 \mathbf{x}]] = \lambda_0 \lambda_1 \text{ReLU}\left[\frac{1}{\lambda_0 \lambda_1} \beta_1 + \Omega_1 \text{ReLU}\left[\frac{1}{\lambda_0} \beta_0 + \Omega_0 \mathbf{x}\right]\right].$$

Explain in one sentence what this says about the uniqueness of a trained network's weights, and connect it to your answer to problem 4 of Week 3.

4. (UDL 4.4) Write out the equations for a network with $D_i = 5$ inputs, $D_o = 4$ outputs, and three hidden layers of sizes $D_1 = 20$, $D_2 = 10$, and $D_3 = 7$, in both the layered form of eq. 14 and the nested form of eq. 15. Give the size of every Ω_k and β_k , and the total parameter count from eq. 16. Then build it with `make_mlp` in the notebook and confirm every size.

5. (UDL 4.7) Choose parameters for the ten-parameter shallow network of Week 3 so that it computes the identity function $y = x$ over a finite range $x \in [a, b]$. Hence explain why the argument of §4.5.1 — that a deep network can approximate any continuous function — is a consequence of the shallow result rather than a new theorem.

6. (UDL 4.8* and 4.9*) Fig. 14 shows three hidden-unit activations with slopes 1, 1, -1 and joints at $x = 1/6$, $2/6$, and $4/6$. Find $\phi_0, \phi_1, \phi_2, \phi_3$ such that $\phi_0 + \phi_1 h_1 + \phi_2 h_2 + \phi_3 h_3$ has four linear regions oscillating between output values zero and one, with the leftmost slope positive. How many regions result if this network is composed with itself? Composed with itself K times? Then decide whether the same kind of oscillation with three regions is possible using two hidden units, and with five regions using four, and explain the pattern.

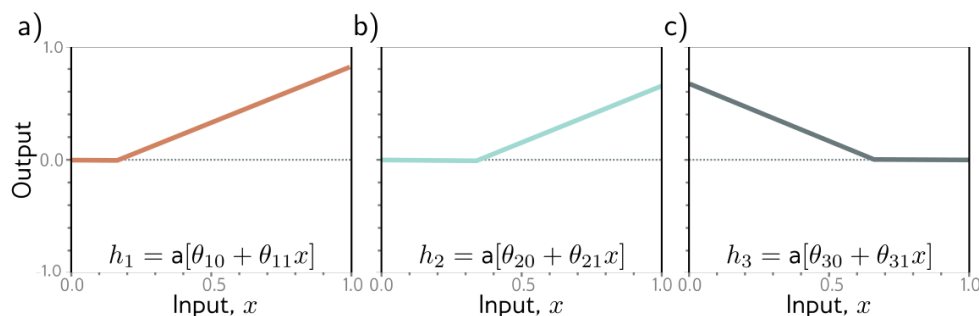


Figure 4.9 Hidden unit activations for problem 4.8. a) First hidden unit has a joint at position $x = 1/6$ and a slope of one in the active region. b) Second hidden unit has a joint at position $x = 2/6$ and a slope of one in the active region. c) Third hidden unit has a joint at position $x = 4/6$ and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

Figure 14: Prince (2024), figure 4.9, p. 55.

7. (UDL 4.10 and 4.11*) Show that a network with one input, one output, and K hidden layers of D units has $3D + 1 + (K - 1)D(D + 1)$ parameters. Then compare the two networks of the PyTorch section: shallow with $D = 95$, and deep with $K = 10$, $D = 5$. Confirm both have 286

parameters, compute the maximum regions for each, and argue which would run faster at inference time. Is the region count the right basis for choosing between them? Use the printed results of the shallow-vs-deep cell in your answer.

8. A laser powder-bed fusion machine records a 256×256 thermal image of the melt pool at each scan step, and a quality model must predict the porosity of the finished part from it. (a) Count the parameters of a fully connected first layer with $D_1 = 512$ hidden units, and comment in the light of §4.5.4. (b) A process engineer describes the physics as a chain: laser parameters set the melt-pool geometry, geometry sets the local cooling rate, cooling rate sets the microstructure, and microstructure sets the porosity. Explain, citing §4.5.2 and the folding picture, why this description is an argument for a deep model rather than a wide one, and write down the hyperparameters of a network whose layers mirror the chain.

9. Computational. The deep network in the PyTorch section is unusually narrow. In the notebook, refit with $K = 3$ layers of $D = 8$ units (169 parameters) and with $K = 2$ layers of $D = 12$, from three different random seeds each. Tabulate final MSE and realized regions for every run. Which configuration is most sensitive to the seed, and what does that suggest about the loss surface that Chapter 6 will have to navigate?

10. Computational. In fig. 5 the region count triples per layer because every region of the zigzag sweeps the full output range $[-1, 1]$. Rescale the first network’s output by 0.5 (halve ϕ_0 and every ϕ_d) and repeat the composition for $K = 1, \dots, 4$. Count the regions, explain why the count now falls short of 3^K , and relate this to the word “up to” in eq. 17.

11 Textbook Reference Map

Every part of this page, mapped to Prince, *Understanding Deep Learning* (MIT Press, 2024) (Prince 2024). Page numbers are those of the printed book.

This page	Prince section	Pages	Figures used
Why Go Deep	Ch. 4 opening	41	—
Composing Two Networks	§4.1	41–43	4.1 (p. 42), 4.2 (p. 44)
— Folding the input space	§4.1	43	4.3 (p. 44)
From Composition to a Deep Network	§4.2	43–45	4.4 (p. 45)
Deep Neural Networks	§4.3	45–46	4.5 (p. 47)
— Hyperparameters	§4.3.1	46	—
Matrix Notation	§4.4	48–49	—
— The general formulation	§4.4.1	48–49	4.6 (p. 48)
Shallow vs. Deep Networks	§4.5	49	—
— Approximation power	§4.5.1	50	—
— Linear regions per parameter	§4.5.2 and Notes	50, 52–53	4.7 (p. 51)
— Depth efficiency	§4.5.3 and Notes	50, 53	—
— Large, structured inputs	§4.5.4	50–51	—

This page	Prince section	Pages	Figures used
— Training and generalization	§4.5.5	51	—
Deep Networks in PyTorch	ours; notebook 4.3	—	—
Summary	§4.6	52	—
Exercises 1–7	Problems 4.1–4.11	53–55	4.8 (p. 54), 4.9 (p. 55)

Equations on this page correspond to Prince as follows: eq. 1 is his (4.1), eq. 2 is (4.2), eq. 3 is (4.3), eq. 4 is (4.4), eq. 5 is (4.5), eq. 6 is (4.6), eq. 8 is (4.7), eq. 9 is (4.8), eq. 10 is (4.9), eq. 11 is (4.10), eq. 12 collects (4.11)–(4.13), eq. 13 is (4.14), eq. 14 is (4.15), eq. 15 is (4.16), and eq. 18 is (4.17) from the chapter notes. Eq. 7 is the outer-product constraint stated in prose on p. 45, eq. 17 is the region and parameter count stated in prose in §4.5.2, and eq. 16 is the general parameter count implied by §4.4.1; none of these three appears as a numbered equation in the book.

i Figure credits

Figures 4.1–4.9 are reproduced unaltered from Prince, *Understanding Deep Learning* (MIT Press, 2024), which is distributed under a Creative Commons CC-BY-NC-ND license, and are used here for non-commercial classroom instruction. All other figures on this page are generated by the code in this week’s notebook.

12 Further Reading

- Prince, *Understanding Deep Learning*, Chapter 4 (Prince 2024) — the assigned reading. The chapter notes (pp. 52–53) carry the history of the term “deep learning”, the region-counting literature, and the depth- and width-efficiency results summarized above.
- **UDL notebooks 4.1–4.3** — Prince’s own Colab notebooks for this chapter; links to each are at the top of this page.
- **This week’s notebook** — every cell above, in one Colab file.
- Montúfar et al. (2014) (Montúfar et al. 2014) — the folding argument and the $(D+1)^K$ style bounds behind eq. 17 and eq. 18.
- Telgarsky (2016) (Telgarsky 2016) — the cleanest depth-efficiency separation: a one-input function that shallow networks need exponential width to match.
- Lu et al. (2017) (Lu et al. 2017) — the depth version of the universal approximation theorem, and the width-efficiency result that makes width the cheaper resource.
- Krizhevsky, Sutskever & Hinton (2012) (Krizhevsky, Sutskever, and Hinton 2012) — the image-classification result that, per the chapter notes, started the modern era of deep learning.
- Goodfellow, Bengio & Courville, *Deep Learning*, §6.4 (Goodfellow, Bengio, and Courville 2016) — a second treatment of architecture design, universal approximation, and the case for depth.

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press. <https://www.deeplearningbook.org/>.

- Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. 2012. “ImageNet Classification with Deep Convolutional Neural Networks.” In *Advances in Neural Information Processing Systems*. Vol. 25.
- Lu, Zhou, Hongming Pu, Feicheng Wang, Zhiqiang Hu, and Liwei Wang. 2017. “The Expressive Power of Neural Networks: A View from the Width.” In *Advances in Neural Information Processing Systems*. Vol. 30.
- Montúfar, Guido, Razvan Pascanu, Kyunghyun Cho, and Yoshua Bengio. 2014. “On the Number of Linear Regions of Deep Neural Networks.” In *Advances in Neural Information Processing Systems*. Vol. 27.
- Prince, Simon J. D. 2024. *Understanding Deep Learning*. MIT Press. <https://udlbook.github.io/udlbook/>.
- Telgarsky, Matus. 2016. “Benefits of Depth in Neural Networks.” In *Conference on Learning Theory*, 1517–39.