but for the moment, we note that it will generally be better to use activation functions with non-zero gradients, at least when the input takes a large positive value. One such choice is the *softplus activation function* given by

Exercise 6.7

$$h(a) = \ln\left(1 + \exp(a)\right), \tag{6.16}$$

which is plotted in Figure 6.12(c). For $a \gg 1$, we have $h(a) \simeq a$, and so the gradient remains non-zero even when the input to the activation function is large and positive, thereby helping to alleviate the vanishing gradients problem.

An even simpler choice of activation function is the *rectified linear unit* or *ReLU*, which is defined by

$$h(a) = \max(0, a) \tag{6.17}$$

and which is plotted in Figure 6.12(d). Empirically, this is one of the best-performing activation functions, and it is in widespread use. Note that strictly speaking, the derivative of the ReLU function is not defined when $a = 0$, but in practice this can be safely ignored. The softplus function (6.16) can be viewed as a smoothed version of the ReLU and is therefore also sometimes called *soft ReLU*.

Exercise 6.5

Although the ReLU has a non-zero gradient for positive input values, this is not the case for negative inputs, which can mean that some hidden units receive no 'error signal' during training. A modification of ReLU that seeks to avoid this issue is called a *leaky ReLU* and is defined by

$$h(a) = \max(0, a) + \alpha \min(0, a), \tag{6.18}$$

where $0 < \alpha < 1$. This function is plotted in Figure 6.12(e). Unlike ReLU, this has a nonzero gradient for input values $a < 0$, which ensures that there is a signal to drive training. A variant of this activation function uses $\alpha = -1$, in which case $h(a) = |a|$, which is plotted in Figure 6.12(f). Another variant allows each hidden unit to have its own value α_j, which can be learned during network training by evaluating gradients with respect to the $\{\alpha_j\}$ along with the gradients with respect to the weights and biases.

The introduction of ReLU gave a big improvement in training efficiency over previous sigmoidal activation functions (Krizhevsky, Sutskever, and Hinton, 2012). As well as allowing deeper networks to be trained efficiently, it is much less sensitive to the random initialization of the weights. It is also well suited to a low-precision implementation, such as 8-bit fixed versus 64-bit floating point, and it is computationally cheap to evaluate. Many practical applications simply use ReLU units as the default unless the goal is explicitly to explore the effects of different choices of activation function.

6.2.4 Weight-space symmetries

One property of feed-forward networks is that multiple distinct choices for the weight vector $\mathbf{w}$ can all give rise to the same mapping function from inputs to outputs (Chen, Lu, and Hecht-Nielsen, 1993). Consider a two-layer network of the form shown in Figure 6.9 with M hidden units having tanh activation functions and full connectivity in both layers. If we change the sign of all the weights and the bias