appropriate cost function for modeling binary data. More appropriate approaches are described in section 6.2.2.2.

Evaluated on our whole training set, the MSE loss function is

$$J(\boldsymbol{\theta}) = \frac{1}{4} \sum_{\boldsymbol{x} \in \mathbb{X}} \left(f^*(\boldsymbol{x}) - f(\boldsymbol{x}; \boldsymbol{\theta})\right)^2 . \tag{6.1}$$

Now we must choose the form of our model, $f(\boldsymbol{x}; \boldsymbol{\theta})$. Suppose that we choose a linear model, with $\boldsymbol{\theta}$ consisting of $\boldsymbol{w}$ and b. Our model is defined to be

$$f(\boldsymbol{x}; \boldsymbol{w}, b) = \boldsymbol{x}^\top \boldsymbol{w} + b. \tag{6.2}$$

We can minimize $J(\boldsymbol{\theta})$ in closed form with respect to $\boldsymbol{w}$ and b using the normal equations.

After solving the normal equations, we obtain $\boldsymbol{w} = \boldsymbol{0}$ and $b = \frac{1}{2}$. The linear model simply outputs 0.5 everywhere. Why does this happen? Figure 6.1 shows how a linear model is not able to represent the XOR function. One way to solve this problem is to use a model that learns a different feature space in which a linear model is able to represent the solution.

Specifically, we will introduce a very simple feedforward network with one hidden layer containing two hidden units. See figure 6.2 for an illustration of this model. This feedforward network has a vector of hidden units $\boldsymbol{h}$ that are computed by a function $f^{(1)}(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{c})$. The values of these hidden units are then used as the input for a second layer. The second layer is the output layer of the network. The output layer is still just a linear regression model, but now it is applied to $\boldsymbol{h}$ rather than to $\boldsymbol{x}$. The network now contains two functions chained together: $\boldsymbol{h} = f^{(1)}(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{c})$ and $y = f^{(2)}(\boldsymbol{h}; \boldsymbol{w}, b)$, with the complete model being $f(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{c}, \boldsymbol{w}, b) = f^{(2)}(f^{(1)}(\boldsymbol{x}))$.

What function should $f^{(1)}$ compute? Linear models have served us well so far, and it may be tempting to make $f^{(1)}$ be linear as well. Unfortunately, if $f^{(1)}$ were linear, then the feedforward network as a whole would remain a linear function of its input. Ignoring the intercept terms for the moment, suppose $f^{(1)}(\boldsymbol{x}) = \boldsymbol{W}^\top \boldsymbol{x}$ and $f^{(2)}(\boldsymbol{h}) = \boldsymbol{h}^\top \boldsymbol{w}$. Then $f(\boldsymbol{x}) = \boldsymbol{w}^\top \boldsymbol{W}^\top \boldsymbol{x}$. We could represent this function as $f(\boldsymbol{x}) = \boldsymbol{x}^\top \boldsymbol{w}'$ where $\boldsymbol{w}' = \boldsymbol{W} \boldsymbol{w}$.

Clearly, we must use a nonlinear function to describe the features. Most neural networks do so using an affine transformation controlled by learned parameters, followed by a fixed, nonlinear function called an activation function. We use that strategy here, by defining $\boldsymbol{h} = g(\boldsymbol{W}^\top \boldsymbol{x} + \boldsymbol{c})$, where $\boldsymbol{W}$ provides the weights of a linear transformation and $\boldsymbol{c}$ the biases. Previously, to describe a linear regression