software environments will raise exceptions when this occurs, others will return a result with a placeholder not-a-number value) or taking the logarithm of zero (this is usually treated as $-\infty$, which then becomes not-a-number if it is used for many further arithmetic operations).

Another highly damaging form of numerical error is **overflow**. Overflow occurs when numbers with large magnitude are approximated as ∞ or $-\infty$. Further arithmetic will usually change these infinite values into not-a-number values.

One example of a function that must be stabilized against underflow and overflow is the softmax function. The softmax function is often used to predict the probabilities associated with a multinoulli distribution. The softmax function is defined to be

$$\text{softmax}(\boldsymbol{x})_i = \frac{\exp(x_i)}{\sum_{j=1}^{n} \exp(x_j)}. \tag{4.1}$$

Consider what happens when all of the x_i are equal to some constant c. Analytically, we can see that all of the outputs should be equal to $\frac{1}{n}$. Numerically, this may not occur when c has large magnitude. If c is very negative, then $\exp(c)$ will underflow. This means the denominator of the softmax will become 0, so the final result is undefined. When c is very large and positive, $\exp(c)$ will overflow, again resulting in the expression as a whole being undefined. Both of these difficulties can be resolved by instead evaluating $\text{softmax}(\boldsymbol{z})$ where $\boldsymbol{z} = \boldsymbol{x} - \max_i x_i$. Simple algebra shows that the value of the softmax function is not changed analytically by adding or subtracting a scalar from the input vector. Subtracting $\max_i x_i$ results in the largest argument to exp being 0, which rules out the possibility of overflow. Likewise, at least one term in the denominator has a value of 1, which rules out the possibility of underflow in the denominator leading to a division by zero.

There is still one small problem. Underflow in the numerator can still cause the expression as a whole to evaluate to zero. This means that if we implement $\log \text{softmax}(\boldsymbol{x})$ by first running the softmax subroutine then passing the result to the log function, we could erroneously obtain $-\infty$. Instead, we must implement a separate function that calculates $\log \text{softmax}$ in a numerically stable way. The $\log \text{softmax}$ function can be stabilized using the same trick as we used to stabilize the softmax function.

For the most part, we do not explicitly detail all of the numerical considerations involved in implementing the various algorithms described in this book. Developers of low-level libraries should keep numerical issues in mind when implementing deep learning algorithms. Most readers of this book can simply rely on low-level libraries that provide stable implementations. In some cases, it is possible to implement a new algorithm and have the new implementation automatically