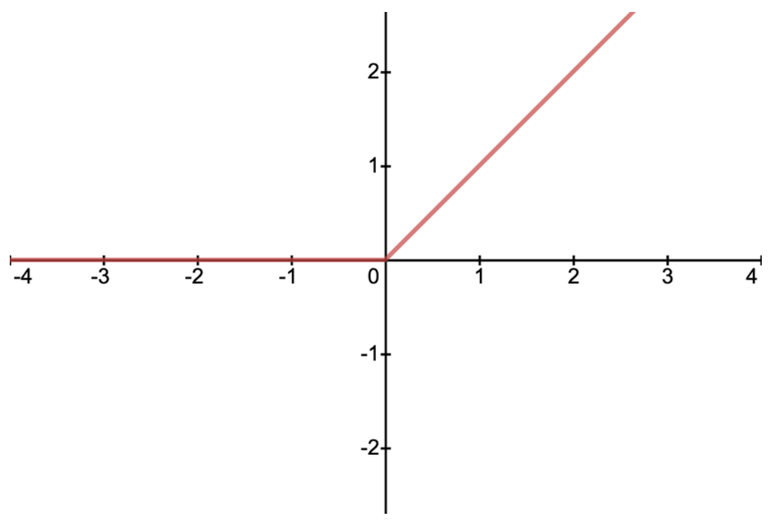
Formalize Hinge Functions

Recall from the video that a neural network is intuitively a sum of many little hinges. Let's write them down concretely.

From the previous module, you know that a hinge can be expressed using $\mathbf{max}(x, 0)$.



However, this function is rather restrictive since the hinge always occurs at $(0, 0)$, which may not be what we want. Rather, we want the hinge to occur at any point in the space (x_0, y_0). So, we can express any sort of hinge using the general version $\mathbf{max}(wx + b, 0) + c$, where $(x_0, y_0) = \left(\frac{-b}{w}, c\right)$.

When the input points x is high-dimensional (vector $\mathbf{x}$), the hinge function can be generalized to $\mathbf{max}\left(\mathbf{w}^\top \mathbf{x} + b, 0\right) + c$.

Thus, a simple one layer neural network with M hinge functions that outputs a scalar $h(\mathbf{x})$ is just a sum of these hinges weighted differently, i.e.,

$$h(\mathbf{x}) = \sum_{i=1}^{M} u_i \left(\mathbf{max}\left(\mathbf{w}_i^\top \mathbf{x} + b_i, 0\right) + c_i\right) = \sum_{i=1}^{M} \left[u_i \, \mathbf{max}\left(\mathbf{w}_i^\top \mathbf{x} + b_i, 0\right) + u_i c_i\right]$$

Or in matrix/vector format:

$$h(\mathbf{x}) = \mathbf{u}^\top \mathbf{max}(\mathbf{W}\mathbf{x} + \mathbf{b}, 0) + c$$

where we stack all the weights $\mathbf{w}_i$ as row vectors vertically to form $\mathbf{W} \in \mathbb{R}^{M \times d}$, collect u_i in a vector $\mathbf{u} \in \mathbb{R}^M$, and define $c = \sum_{i=1}^{M} u_i c_i$. To clarify, $\mathbf{max}(\cdot, 0)$ is an element-wise max function.

If we want multi-dimensional outputs (a vector $\mathbf{h}(\mathbf{x}) \in \mathbb{R}^N$), we can simply change the vector $\mathbf{u}$ to a matrix $\mathbf{U} \in \mathbb{R}^{N \times M}$ and the scalar c to a vector $\mathbf{c} \in \mathbb{R}^N$, yielding $\mathbf{h}(\mathbf{x}) = \mathbf{U} \, \mathbf{max}(\mathbf{W}\mathbf{x} + \mathbf{b}, 0) + \mathbf{c}$.

Note that $\mathbf{U}, \mathbf{W}, \mathbf{b}$ and $\mathbf{c}$ are all parameters that we are going to learn from the data. The most common way to learn these parameters is using Stochastic Gradient Descent, which is a variant of gradient descent. We will discuss SGD later.

To summarize, a neural network can approximate decision boundary locally using kinks. The direction and position of those kinks are determined by the parameters $\mathbf{U}, \mathbf{W}, \mathbf{b}, \mathbf{c}$ that are learned from data using Stochastic Gradient Descent. If you introduce enough kinks, you can approximate any continuous function arbitrarily well.

☆ **Key Points**

The parameters being used are $\mathbf{U}, \mathbf{W}, \mathbf{b}$ and $\mathbf{c}$. The parameters are learned from the data.

Stochastic gradient descent (SGD) is the most common way to learn the parameter.