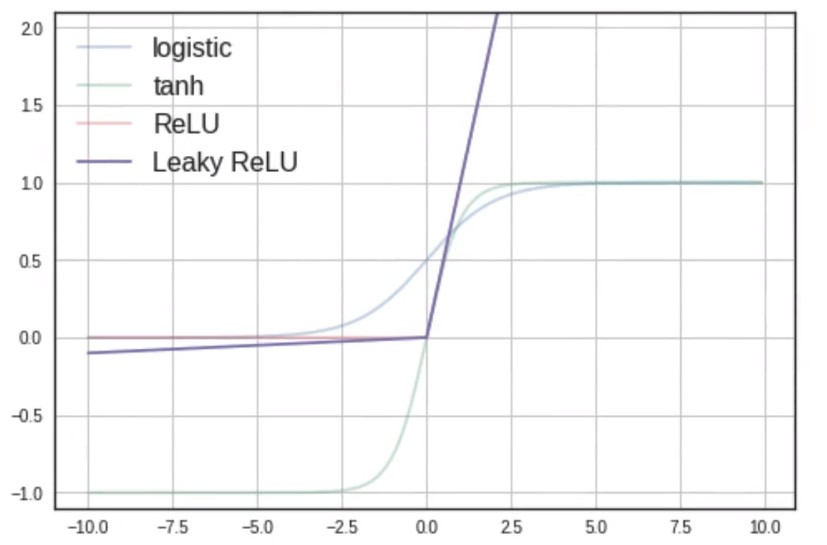
## **Tanh and ReLU Activation Functions**

Is there any caveat in using ReLU?

1. Consider the following deep neural network that uses the ReLU activation function 
   1. What happens if b takes on a large negative value due to a large negative update at some point?
   2. [if b << 0]
   3. Therefore [dead neuron]
   4. Which means
   5. This zero derivative is involved in the chain rule for computing the gradient w.r.t
   6. leading to the weight not being updated, as in the case of a **saturated neuron**.
   7. This also applies to and , their parameters are not updated.
   8. Here, x1 and x2 have been normalised, so they range between 0-1 and are therefore unable to counterbalance any large negative value b
   9. This means that once a neuron has died, it remains dead forever, as no new input would be large enough to counter the negative b value
   10. Thus, there is a very real problem of saturation of a ReLU neuron in the negative region.
   11. In practice, if there is a large number of ReLU neurons, a large fraction (up to 50%) may die during operation if the learning rate is set too high
   12. It is advised to initialise the bias to a positive value
   13. Using other variants of ReLU is recommended
2. A good alternative is the **Leaky ReLU**
   1. The following figure illustrates the leaky ReLU function
   2. ReLU outputs the input value itself if it is positive, else it outputs a fraction of the input value, i.e. f(2) = 2, f(-2) = 0.02
   3. It does not saturate in the positive or negative region
   4. Will not die (0.01x ensures that at least a small gradient will flow through), this means that there isn’t any 0 valued derivative, thereby ensuring that the gradients are all non-zero. Thus, the weights are always updated.
   5. It is easy to compute (no expensive ex)
   6. Close to zero centered outputs