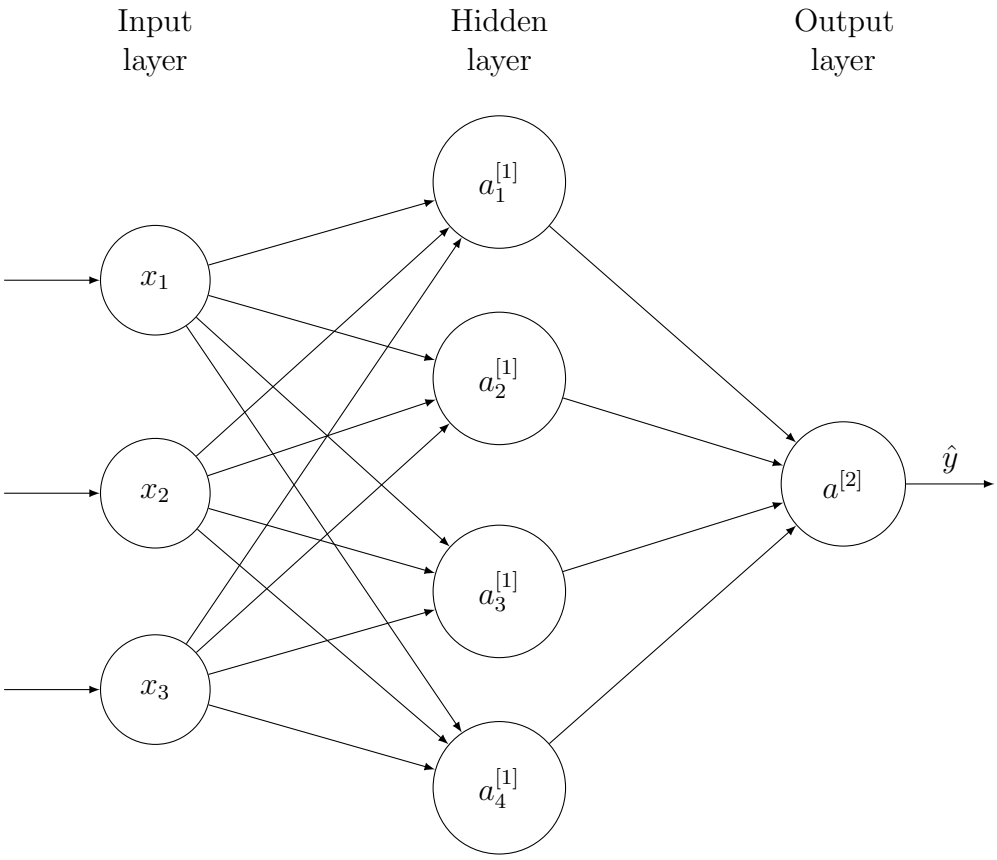
Neural Network Representation

The following diagram represents a two layer Neural Network:



In this case, two parameters are associated with the hidden layer:

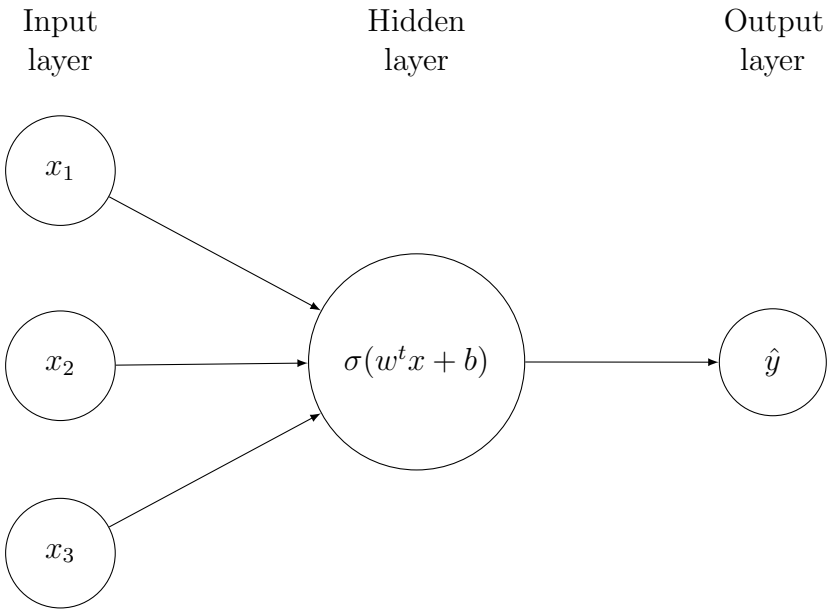
$$W^{[1]} \in \mathbb{R}^{4\times 3}$$
$$b^{[1]} \in \mathbb{R}^{4\times 1}$$

Similarly two parameters are associated with the output layer:

$$W^{[2]} \in \mathbb{R}^{1\times 4}$$
$$b^{[2]} \in \mathbb{R}$$

Computing a Neural Network's Output (one input)

Remember that for logistic regresion we did the following computation



Another way to think about a neural network is that that each hidden layer is performing a logistic regression. Each of the logistic regresion units has it's own w and b parameters. So, in a neural network, each neuron does the following computation:

$$
\begin{aligned}
z_i^{[1]} &= w_i^{[1]} x + b_i^{[1]} \\
a_i^{[1]} &= \sigma(z_i^{[1]}) \\
z_i^{[2]} &= w_i^{[2]} a_i^{[1]} + b_i^{[2]} \\
a_i^{[2]} &= \sigma(z_i^{[2]})
\end{aligned}
$$

Or in matrix notation

$$
\begin{aligned}
z^{[1]} &= W^{[1]}x + b^{[1]} \\
a^{[1]} &= \sigma(z^{[1]}) \\
z^{[2]} &= W^{[1]}a^{[1]} + b^{[2]} \\
a^{[2]} &= \sigma(z^{[2]})
\end{aligned}
$$

Computing a Neural Network's Output (multiple inputs)

We define the following matrices:

$$
\begin{aligned}
X &= \left[x^{(1)} \mid \cdots \mid x^{(m)}\right] \\
Z^{[i]} &= \left[z^{[i](1)} \mid \cdots \mid z^{[i](m)}\right] \\
A^{[i]} &= \left[a^{[i](1)} \mid \cdots \mid a^{[i](m)}\right]
\end{aligned}
$$

And then, the vectorized form for the forward propagation calculations are:

$$
\begin{aligned}
Z^{[1]} &= W^{[1]}X + b^{[1]} \\
A^{[1]} &= \sigma(Z^{[1]}) \\
Z^{[2]} &= W^{[2]}A^{[1]} + b^{[2]} \\
A^{[2]} &= \sigma(Z^{[2]})
\end{aligned}
$$

Activation functions

We replace the sigmoid function σ with a more general function g

$$A^{[i]} = \cancel{\sigma(Z^{[i]})}\ g(Z^{[i]})$$

Common activation functions are:

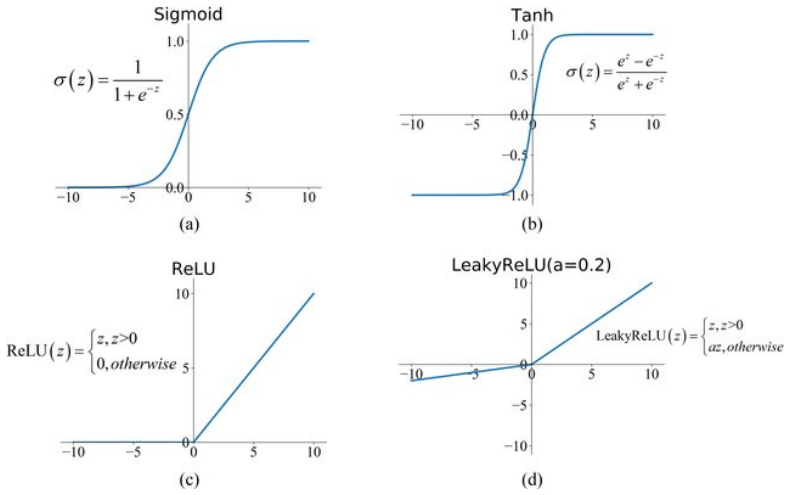


Figure 1: Common activation functions

If your output is binary the sigmoid is ideal for the output layer. In any other case the hyperbolic tangent activation function is superior than the sigmoid.

The most common activation function is the ReLU and it usually performs better because for a lot of the space of Z, the derivative of the activation function if far from 0. So, using the ReLU activation function, the neural network will often learn much faster than when using the tanh or the sigmoid activation functions.

One disadvantage of ReLU is that for negative x the derivative is zero, it works fine in practice but that can be fixed using the leaky ReLU.

Finally, to implement back propagation for a neural network, it's necessary to compute the derivative of the activation functions, the following table summarises the derivatives of the former activation functions.

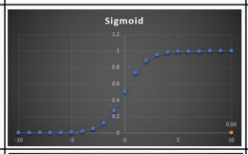
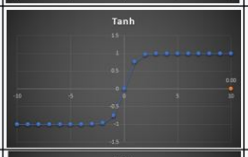
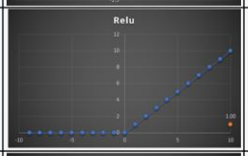
Name	Plot	Equation	Derivative
Sigmoid		$f(x) = \sigma(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1-f(x))$
Tanh		$f(x) = \tanh(x) = \frac{(e^x - e^{-x})}{(e^x + e^{-x})}$	$f'(x) = 1 - f(x)^2$
Rectified Linear Unit (relu)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Leaky Rectified Linear Unit (Leaky relu)		$f(x) = \begin{cases} 0.01x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0.01 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$

Figure 2: Derivatives of activation functions

Importance of non-linear activation functions

Since the composition of linear functions is linear, using linear activation functions outputs a linear function of the input and you might as well not have any hidden layers. The whole purpose of the neural networksis to capture non-linear relationships in the input data and that's why it's important to use non-linear activation functions.

The one place in which you can use a linear activation function is in the output layer of regression problems when your output is a real number.

Gradient descent for Neural Networks

Using te following cost function:

$$J(W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}) = \frac{1}{n} \sum_{i=1}^{n} L(\hat{y}, y)$$

We can compute the derivatives using the computation graph as we did with the logistic function. The derivatives are:

$$
\begin{aligned}
dZ^{[2]} &= A^{[2]} - Y \\
dW^{[2]} &= \frac{1}{m} dZ^{[2]} A^{[1]\top} \\
db^{[2]} &= \frac{1}{n} sum(dZ^2)
\end{aligned}
$$

$$
\begin{aligned}
dZ^{[1]} &= W^{[2]\top} dZ^{[2]} * g^{[1]'}(Z^[1]) \\
dW^{[1]} &= \frac{1}{m} dZ^{[1]} X\top \\
db^{[1]} &= \frac{1}{n} sum(dZ^1)
\end{aligned}
$$

Weight initialization

If the weights are initialized to zero (as we did with the logistic regresion) every hidden unit will be completely identical and, in fact, they will be exactly the same after every iteration, then there's really no point in having several hidden units.

As you want the different hidden units to compute different functions, we usually initialize every weight matrix with small observations of random gaussian variables. The bias terms can be initialized with zeros.