A perceptron is a basic building block of a neural network. It takes in several inputs, weighs them, and then puts the result through an activation function to make a decision.The activation function decides if the perceptron should be "turned on" or not.

### Activation Functions

Activation functions are math formulas that help a neural network learn complex things. They take the output of a perceptron and change it into a form that's easier for the next layer to use. Here are some common ones:

* **ReLU (Rectified Linear Unit):** This one is very simple. If the input is positive, it keeps it the same.6 If the input is zero or negative, it changes it to zero.
* **Sigmoid:** This function squishes any input value into a number between 0 and 1. It's often used when you need to predict a probability.
* **Softmax:** This is used when you need to pick one answer from a list of many. It takes a bunch of numbers and turns them into probabilities that add up to 1. The biggest number in the original list will get the highest probability.

### Backpropagation & Gradient Descent

These are two key ideas for training neural networks.

**Gradient Descent** is a way to find the lowest point on a "valley" of errors.Imagine you're on a mountain and want to get to the lowest point. You can't see the whole mountain, but you can feel the slope right where you are. You take a step in the steepest downward direction. You keep doing this until you get to the bottom. In a neural network, the "mountain" is the error, and the "steps" are changes to the perceptron's weights. Gradient descent helps the network slowly find the best weights to make its predictions more accurate.

**Backpropagation** is the method that actually calculates the "steepest downward direction" for each weight. It works by going backward through the network, from the output layer to the input layer. It looks at the difference between the network's prediction and the correct answer. Then, it uses that difference to figure out how much each weight in the network contributed to the error. This information is then used by gradient descent to update the weights.

Think of it like this:

1. **Forward Pass:** The network makes a guess.
2. **Calculate Error:** You find out how wrong the guess was.
3. **Backpropagation:** You trace the error backward through the network to see which weights were most responsible.
4. **Gradient Descent:** You use the information from backpropagation to slightly adjust the weights, so the next guess will be a little better.

This process is repeated many times until the network gets really good at what it's supposed to do.