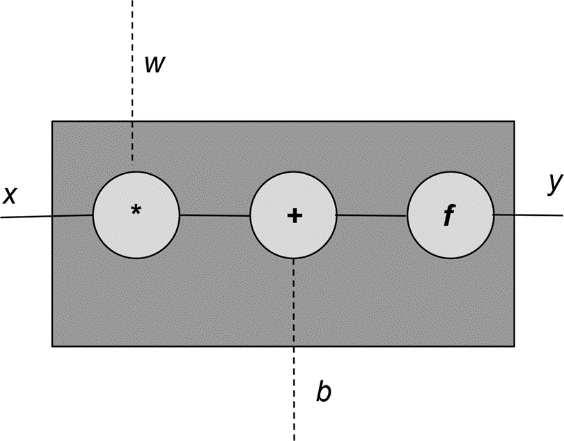
**Foundational Components of Neural Networks**

**The perceptron:**



Each perceptron unit has an input (x), an output (y), and three “knobs”: a set of weights (w), a bias (b), and an activation function (f). The weights and the bias are learned from the data, and the activation function is handpicked depending on the network designer’s intuition of the network and its target outputs. Mathematically, we can express this as follows:

y = f ( wx + b )

It is usually the case that there is more than one input to the perceptron. We can represent this general case using vectors. That is, x, and w are vectors, and the product of w and x is replaced with a dot product:

y = f ( wx + b )

The activation function, denoted here by f, is typically a nonlinear function. A linear function is one whose graph is a straight line. In this example, wx+b is a linear function. So, essentially, a perceptron is a composition of a linear and a nonlinear function. The linear expression wx+b is also known as an affine transform.

Code:

|  |
| --- |
| import torch  import torch.nn as nn  class Perceptron(nn.Module):  """ A perceptron is one linear layer """  def \_\_init\_\_(self, input\_dim):  """  Args:  input\_dim (int): size of the input features  """  super(Perceptron, self).\_\_init\_\_()  self.fc1 = nn.Linear(input\_dim, 1)    def forward(self, x\_in):  """The forward pass of the perceptron    Args:  x\_in (torch.Tensor): an input data tensor  x\_in.shape should be (batch, num\_features)  Returns:  the resulting tensor. tensor.shape should be (batch,).  """  return torch.sigmoid(self.fc1(x\_in)).squeeze() |

**Activation Functions:**

1. **Sigmoid:**
2. **Tanh**
3. **ReLU**
4. **Softmax**

**Loss Functions:**

1. **Mean Squared Error Loss**
2. **Categorical Cross-Entropy Loss**
3. **Binary Cross-Entropy Loss**