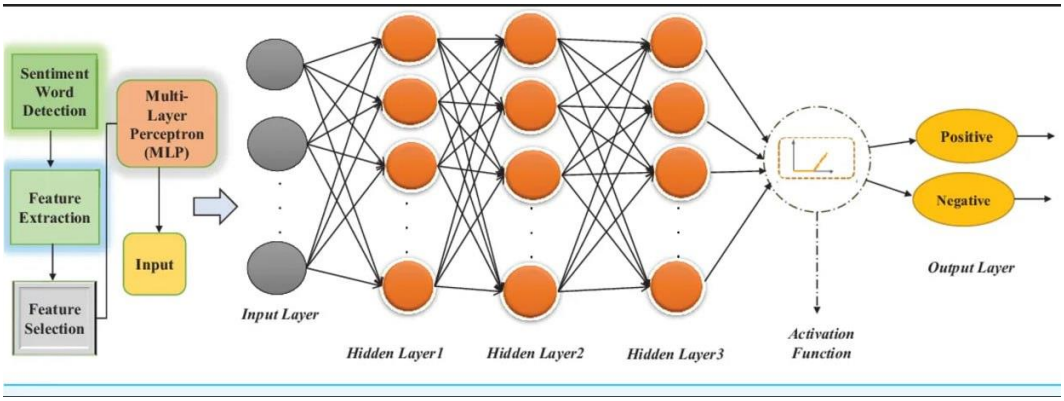
Multilayer Perceptron

1. Draw of architecture model.



2. Vector representation of data:
   Input (X):
   [[0 0]
   [0 1]
   [1 0]
   [1 1]]
   Input (Y): [1 1 0 1]
   Output:
   Weights for hidden layers: [[-4.59946067 4.69624527]
   [-2.32246687 2.50974137]]
   Bias for hidden layers: [1.84450874 0.41237148]
   Weights for output layers: [[7.66399363 2.89578143]]
   Bias for output layers: [-4.34133134]
   Loss = 0.00017
   Output preds = [0.985658]
   Accuracy = 100.0 %
3. Math formulation of linear combination, activation function and loss function.
   Linear combination:

$$z^{(l)} = W^{(l)} * a^{(l-1)} + b^{(l)}$$

   Activation function:

$$a^{(l)} = \sigma(z^{(l)})$$

   Loss function:

$$\delta^{(l)} = \frac{\partial L}{\partial z^{(l)}} = \delta^{(l+1)} * W^{(l+1)} * b'(z^{(l)})$$

4. Math formulation of how neural nets calculate the predictions (y_hat).

$$\hat{y} = \sigma(W^{(2)} h^{(1)} + b^{(2)})$$

1. Explanation of gradient descendent algorithm and formulas of gradients and weights/biases updates.

   Multilayer perceptron (MLP) is trained by gradient descent, which consists of three steps:

   1. Forward propagation:

$$z_i^j = w_{ij}^T x^{j-1} + b_i^j \; a_i^j = \sigma(z_i^j)$$

where z_i^j is the weighted length, a_i^j is the activation function (σ), w_{ij} is the weight, b_{ij} is the bias, x^{j-1} is the layer output.

2. Backpropagation:

The gradients compete using the chain rule:

$$\partial L/\partial w_{ij} = \partial L/\partial a_{ij} * \partial a_{ij}/\partial z_{ij} * \partial z_{ij}/\partial w_{ij} \; \partial L/\partial b_{ij} = \partial L/\partial a_{ij} * \partial a_{ij}/\partial z_{ij} * \partial z_{ij}/\partial b_{ij}$$

where L is the loss function.

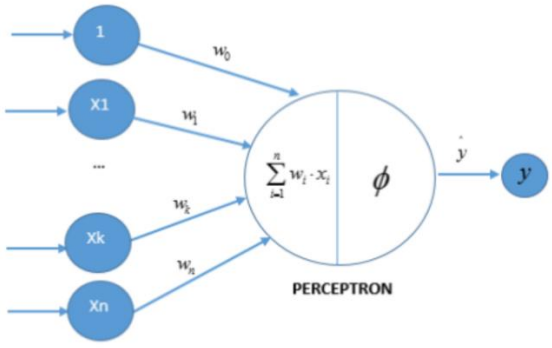
3. Updating weights:

$$w_{ij} = w_{ij} - \eta * \partial L/\partial w_{ij} \; b_{ij} = b_{ij} - \eta * \partial L/\partial b_{ij}$$

where η is the learning rate.

Perceptron

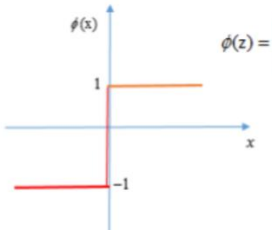
1. Draw of architecture model and math formulation of linear combination, activation function and loss function and math formulation of how neural nets calculate the predictions (y_hat):



$$X = \begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{bmatrix} \quad W = \begin{bmatrix} w_0 \\ w_1 \\ \dots \\ w_n \end{bmatrix}$$

$$z = X^T \cdot W = w_0 + \sum_{i=1}^{n} x_i \cdot w_i$$

$$\phi(z) = \begin{cases} 1, & se\ z \geq 0 \\ -1, & se\ z < 0 \end{cases}$$



$$L(\mathbf{w}, \mathbf{x}, y) = \max(0, -y(\mathbf{w} \cdot \mathbf{x}))$$

where:

- $\mathbf{w}$ is the weight vector.
- $\mathbf{x}$ is the input feature vector.
- y is the true label (-1 or 1).

2. Vector representation of data.

Input data (X): array([[89.59264065, 50.50169383],

[46.48970404, 4.99005293],
[83.07284289, 14.85578394],
[74.36775792, 67.25528206],
[96.85882241, 47.67644428]])

Input data (Y): array([-1, 1, 1, -1, 1, 1, 1, 1, 1, -1, 1, -1, 1, 1, 1, 1, -1,

1, 1, 1, 1, 1, -1, 1, 1, 1, -1, 1, 1, 1, 1, 1, 1, -1,
-1, 1, 1, 1, 1, 1, 1, 1, 1, -1, 1, 1, 1, 1, 1, -1, 1,
1, 1, -1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, -1, 1, 1, -1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1])

Output data: array([1.0943129 , 2.26101084, 0.3637471])

print("Percetron classifier model: ")

print(f"rain = {neuron.w_[0]:.4f} + {neuron.w_[1]:.4f} * Humidity + {neuron.w_[2]:.4f} * Cloud-Cover")

class = 1

3. Explanation of gradient descendent algorithm.
   Gradient descent is a method for finding the minimum of a function (in this case, the error of the function).

$$\frac{dL}{dw} = \frac{dL}{dz} * \frac{dz}{dw}$$

   The gradient is a vector indicating the direction of the steepest increase of the function. For the Hinge Loss function, the gradient over the weight is calculated for each training set variant. If the prediction is incorrect ($y * \hat{y} < 1$), the gradient over the weight w_i is equal to $-y * x_i$. Otherwise, the gradient is 0.
   Weight update: weights are changed by moving in the direction opposite to the gradient (how we want to minimize the errors of the function). The formula for weight update is:

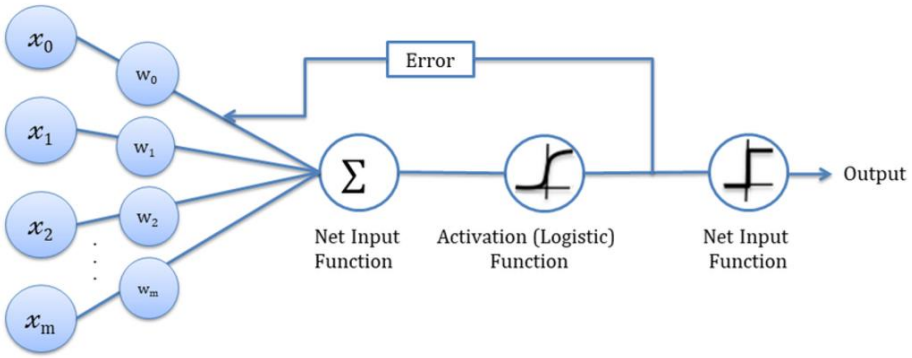
   $w_i = w_i - \eta * \nabla L(w_i)$

   where:

- η— learning rate — a hyperparameter that determines the step size of weight update. Too large a value can led to wastage, too small — to slow convergence.
- $\nabla L(w_i)$— partial derivative of the error function with respect to the weight w_i (gradient component). The process of computing the gradient and updating the weights is repeated iteratively until the function error is minimum or a specified number of iterations is reached.

Logistic Regression

2. Draw of architecture model.



3. Vector representation of data.
   Input (X): [89.59264065 50.50169383]
   [46.48970404 4.99005293]
   [83.07284289 14.85578394]
   [74.36775792 67.25528206]
   [96.85882241 47.67644428]
   Input (Y): [1 0 0 1 0 0 0 0 0 1 0 1 0 0 0 0 1 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 1 1 0 0
   0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
   0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
   Output data:
   ========== Epoch 0 ==========
   Train loss: 11.399511076097532
   Accuracy: 0.86

   ========== Epoch 10 ==========
   Train loss: 15.51622952723511 WARNING - Loss Increasing
   Accuracy: 0.86
4. Math formulation of linear combination, activation function and loss function.
   Sigmoid activation function:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

   Linear combination:

$$\hat{y} = \sigma(\omega_1 x_1 + \omega_2 x_2 + b)$$

   Loss function:

$$L(y, \hat{y}) = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(\hat{y}_i) + (1 + y_i)\log(1 + \hat{y}_i)]$$

5. Explanation of gradient descendent algorithm and formulas of gradients and weights/biases updates.
   Gradient descent is an optimization method used to minimize the loss function in machine learning models. Its goal is to find parameters that reduce the error between predicted and actual values.

   Initialization occurs with given initial values for the parameters (weights).

Gradient calculation: θ:=θ−α∇J(θ)
Parameter update: θj:=θj−(α*1/m)∑ i=1 m (hθ(x^(i))−y^(i))xj^(i)
Where θj is the j-th parameter, and ax (j) (i) is the j-th feature of the i-th training example.