Perceptron, Activation Functions, Backpropagation & Gradient Descent

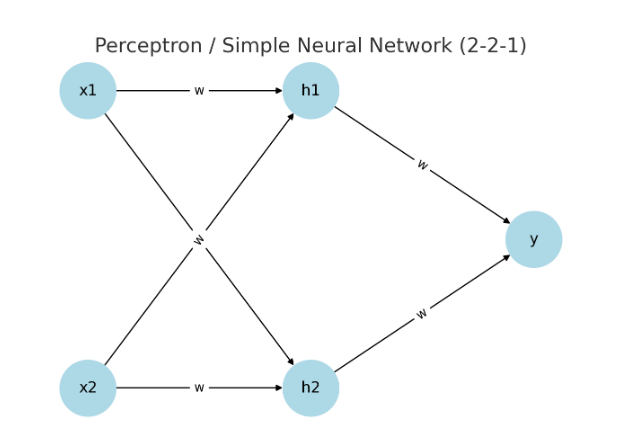
# 1. Perceptron

Concept: A perceptron takes weighted inputs, applies an activation function, and gives an output.

Formula: y = f( Σ (w\_i \* x\_i) + b )

Python Example:

import numpy as np  
  
def perceptron(x, w, b, activation):  
 z = np.dot(x, w) + b  
 return activation(z)  
  
def step(z):  
 return 1 if z >= 0 else 0  
  
x = np.array([1, 0])   
w = np.array([0.5, 0.5])   
b = -0.2   
  
output = perceptron(x, w, b, step)  
print("Perceptron Output:", output)



# 2. Activation Functions

## Sigmoid

Formula: f(x) = 1 / (1 + e^(-x))

def sigmoid(x):  
 return 1 / (1 + np.exp(-x))

## ReLU

Formula: f(x) = max(0, x)

def relu(x):  
 return np.maximum(0, x)

## Softmax

Formula: f(x\_i) = e^(x\_i) / Σ e^(x\_j)

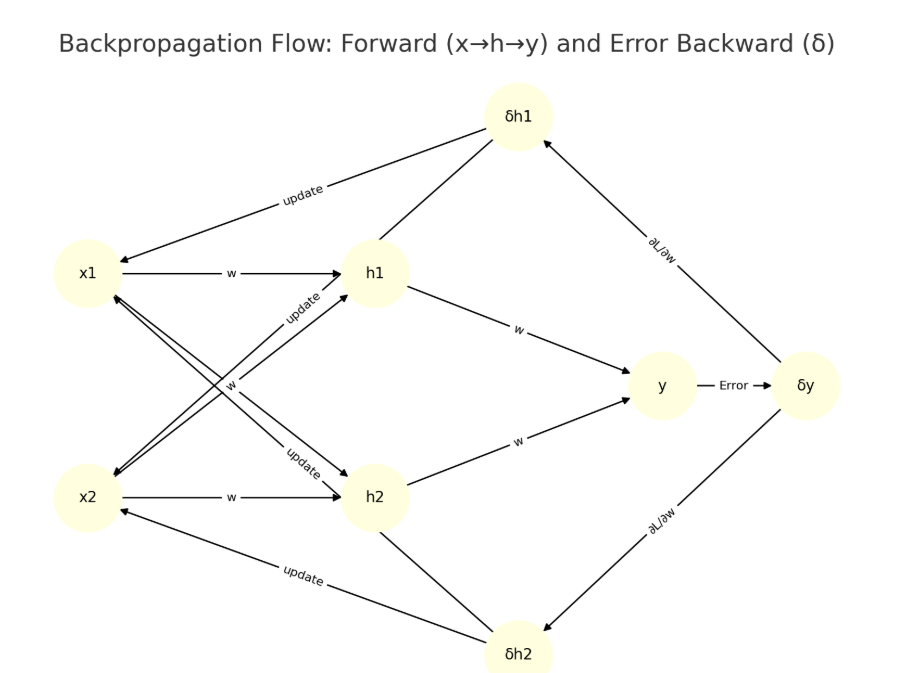
def softmax(x):  
 exps = np.exp(x - np.max(x))  
 return exps / np.sum(exps)

Example:

vals = np.array([-2, 0, 3])  
print("Sigmoid:", sigmoid(vals))  
print("ReLU:", relu(vals))  
print("Softmax:", softmax(vals))

# 3. Backpropagation & Gradient Descent

Forward pass → calculate prediction  
Compute loss → error  
Backpropagation → gradients  
Update weights using Gradient Descent: w = w - η \* (∂L/∂w)



# 4. Simple Neural Network (Backpropagation without Libraries)

import numpy as np  
  
def sigmoid(x):  
 return 1 / (1 + np.exp(-x))  
  
def sigmoid\_derivative(x):  
 return x \* (1 - x)  
  
X = np.array([[0,0],[0,1],[1,0],[1,1]])  
y = np.array([[0],[1],[1],[0]])  
  
np.random.seed(1)  
w1 = np.random.rand(2, 2)  
w2 = np.random.rand(2, 1)  
lr = 0.5  
  
for epoch in range(5000):  
 hidden = sigmoid(np.dot(X, w1))  
 output = sigmoid(np.dot(hidden, w2))  
 error = y - output  
 d\_output = error \* sigmoid\_derivative(output)  
 d\_hidden = d\_output.dot(w2.T) \* sigmoid\_derivative(hidden)  
 w2 += hidden.T.dot(d\_output) \* lr  
 w1 += X.T.dot(d\_hidden) \* lr  
  
print("Final predictions:\n", output)

# 5. TensorFlow/Keras Example

import tensorflow as tf  
from tensorflow.keras import layers, models  
  
X = [[0,0],[0,1],[1,0],[1,1]]  
y = [[0],[1],[1],[0]]  
  
model = models.Sequential([  
 layers.Dense(2, input\_dim=2, activation="sigmoid"),  
 layers.Dense(1, activation="sigmoid")  
])  
  
model.compile(optimizer="adam", loss="binary\_crossentropy", metrics=["accuracy"])  
model.fit(X, y, epochs=500, verbose=0)  
  
print("Predictions:", model.predict(X))