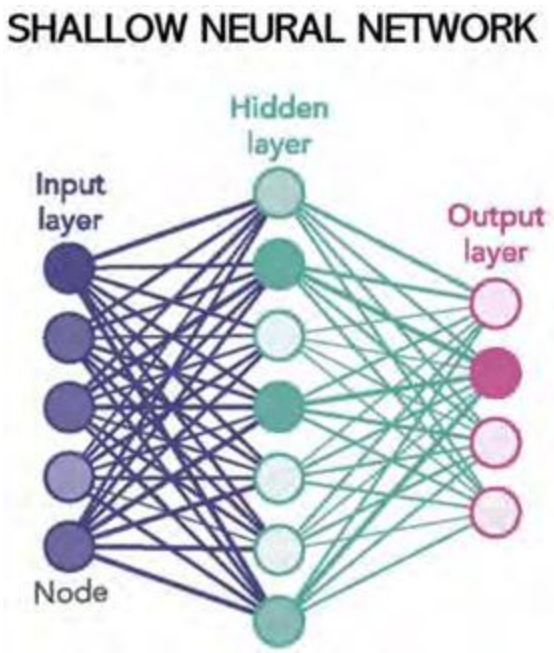
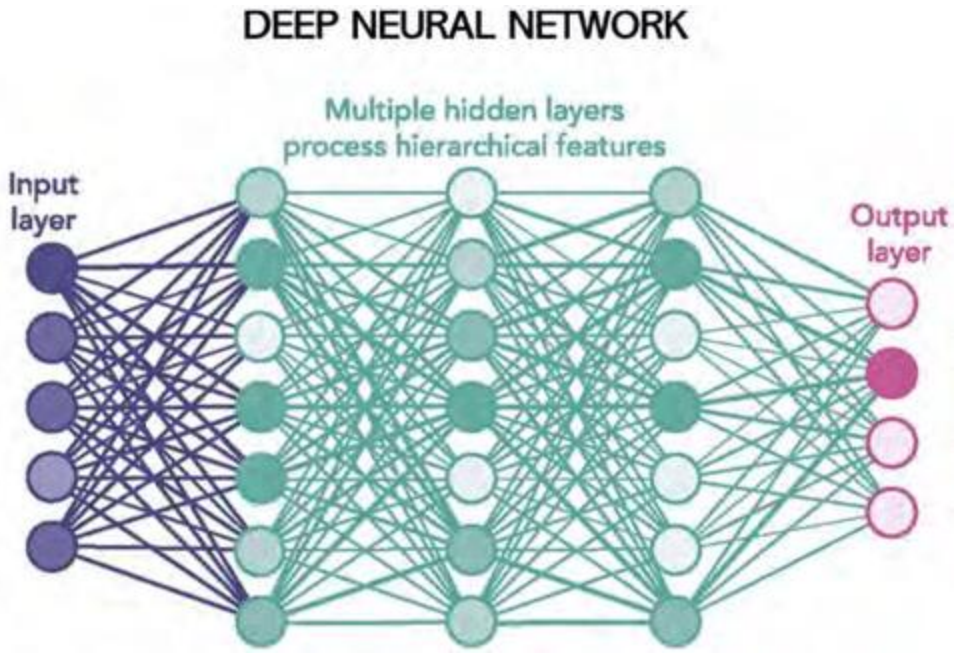
Redes Neuronales y Aprendizaje Profundo

Bloque1: Feedforward Neural Networks

From Shallow to Deep NN





Feedforward Neural Networks

- **Objectives**
  - To understand the fundamentals of a basic multilayer perceptron neural network **MLP/ Shallow neural network**.
  - To understand the concept of a "**basic neuron**" and the concepts of "**activation function**", "**loss function**" and by extension the "**cost function**".
  - To understand the concept of "**gradient descent**" through the use of function **derivatives** and **the chain rule**.

Feedforward Neural Networks

- **Objectives**
  - To understand the idea of **forward and backward propagation** as a basis for the search for minima in the gradient descent process.
  - To analyze the basis of **deep neural networks** with multiple inner layers.
  - Understanding **hyperparameters** in a deep neural network and some techniques for improving error.

Feedforward Neural Networks

- **Contents**
  - Supervised learning fundamentals
  - Perceptron Neuron
  - Activation Function types
  - MLP Multilayer Perceptron
  - Deep L-layer Neural Networks

Supervised learning fundamentals

- **Supervised learning fundamentals**
  - Supervised learning describes tasks where a dataset containing features and labels is provided and the model must predict the labels when given input features.
  - Unsupervised learning models receive unlabeled data to discover patterns without any prior explicit guidance or instruction.

Supervised learning fundamentals

- **Training/test datasets**
  - **The training dataset** is used to train a model by allowing it to learn patterns, relationships, and parameters from the data. The model adjusts its internal weights based on the training dataset, typically larger than the test dataset to ensure the model learns well.
  - **The test dataset** is used to evaluate the model's performance after training. The model has never seen this data before, ensuring an unbiased assessment. Helps determine if the model generalizes well to new, unseen data.
  - Sometimes a **validation set** is used between training and testing to fine-tune hyperparameters (e.g., learning rate, model complexity) and avoid overfitting.

Supervised learning fundamentals

- **Training/generalization error.**
  - **Training error:** The error the model makes on the training dataset, the data it has been directly trained on.
  - **Generalization error:** The error measured on new, unseen data (test or validation data). The generalization error is estimated by applying our model to an independent test set constituted of a random selection of examples.

Supervised learning fundamentals

- **Linear regression & Classification**

  - **Linear regression** is a **supervised learning algorithm** used for predicting a continuous output based on input features.
  - Linear regression problems answer questions such as how much or how many
  - **Classification** problems answer questions such as which category

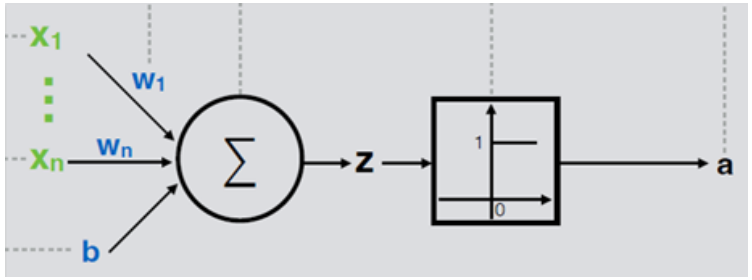
- **Cross validation**

  - The **cross-validation technique** is often used when training data is sparse and we cannot afford to build an adequate validation set.

  - The **original training data is divided into non-overlapping subsets**. Training and model validation are then run multiple times, each of which is trained on subsets and validated on a different subset (the one that was not used for training in that round).

Perceptron Neuron

- **Perceptron neuron. Mathematical model**



$$z = (\sum_{i=1}^{n} x_i * w_i + b); a = \sigma(z)$$

x_i: input characteristics

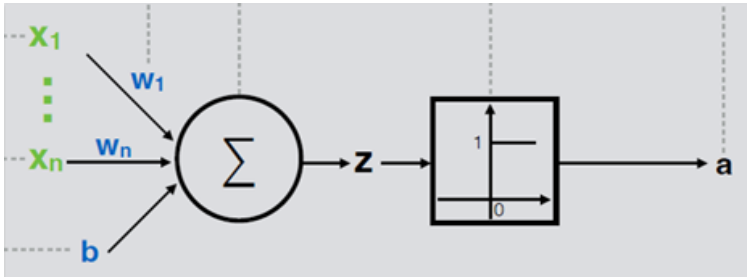
w_i: weights

b: bias

Σ: activation function

Perceptron Neuron

- **Perceptron neuron. Mathematical model**



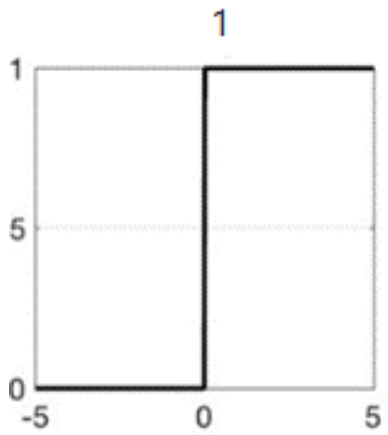
  - What is the **purpose** of **bias "b"**?
  - We need the bias "b" because it allows us to express all the linear functions of our features, **without restricting** ourselves to the **lines passing through the origin**.
  - The bias "b" determines **the value of the estimate when all features are zero.**

Perceptron Neuron

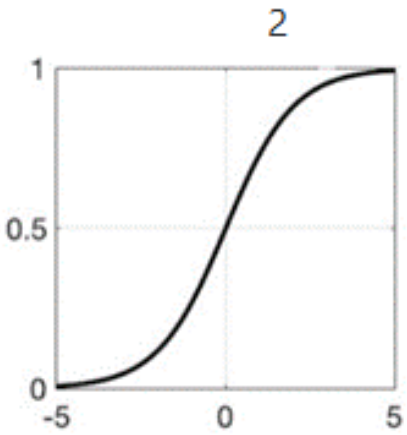
- **Activation Function types**
  - Activation functions **decide whether a neuron should be activated** or not.
  - Activation functions can introduce **nonlinearities** in the outputs of individual units/neurons.
  - They **help the model learn complex patterns** beyond just linear relationships.

Perceptron Neuron

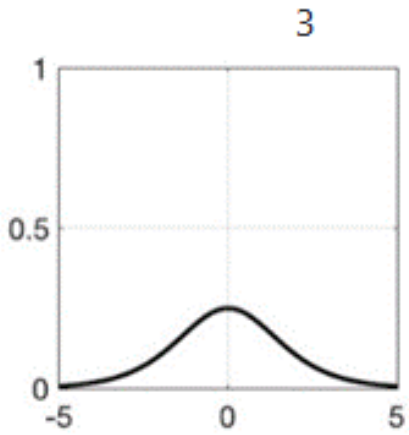
- **Common activation functions**



Step

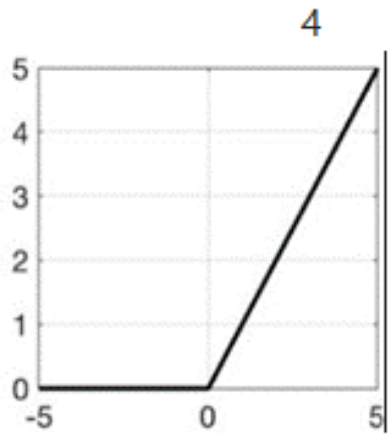


Sigmoid



Sigmoid

(derivative)



ReLu

Perceptron Neuron

- **Common activation functions expressions**

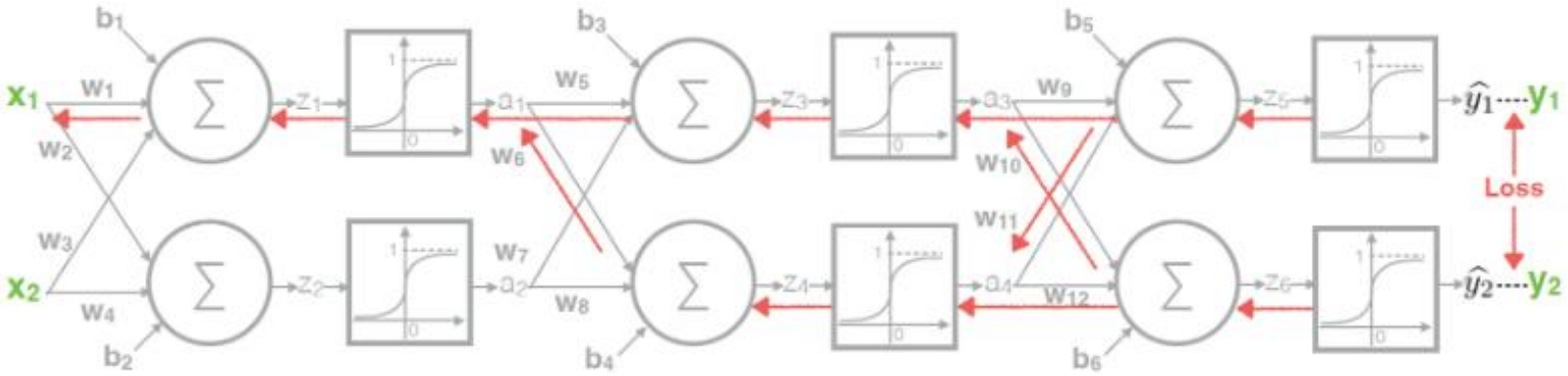
1: $f(x) = \max(x, 0);$ **ReLu**

2: $f(x) = \frac{1}{1+e^{-x}};$ **Sigmoid**

3: $f(x) = \max(x, 0) + \alpha * \min(0, x);$ **pReLU (parameterized ReLU)**

MLP Multilayer Perceptron

- **Shallow neural networks**



Deep L-layer Neural Networks

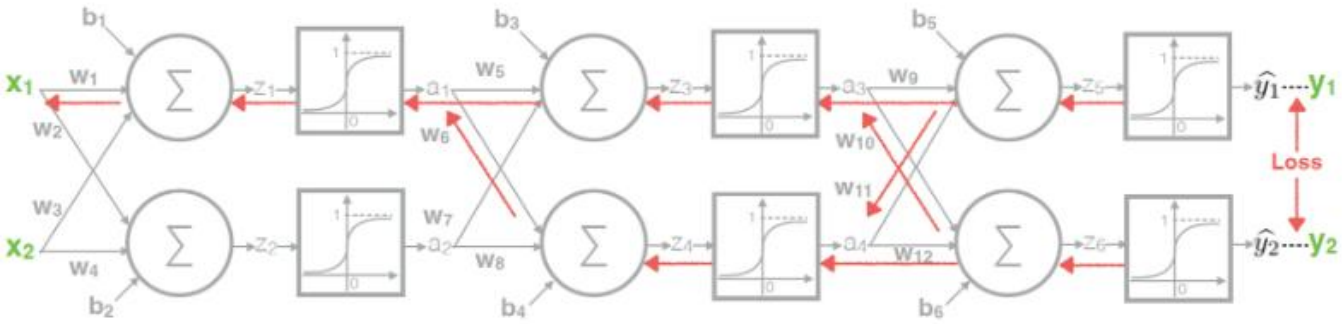
- **Shallow neural networks**
  - **Shallow Neural Network:** Contains only one hidden layer
  - **Deep Neural Network:** Contains two or more hidden layers
    - A **single-layer perceptron** can **only** model **linear relationships**.
    - A single hidden layer can approximate any continuous function but **adding more layers helps with feature extraction and abstraction**.
    - **Deep networks** (**2+ hidden layers**) **allow learning of complex hierarchical patterns**, making them more powerful for tasks like image recognition and natural language processing.

Deep L-layer Neural Networks

- **Hyperparameters**
  - **Constant parameters** whose **values** are **fixed before the learning process**
    - Number of layers
    - Number of neurons per layer
    - Type of activation function
    - Type of loss function
    - Optimization method
    - Learning rate and
    - Batch size

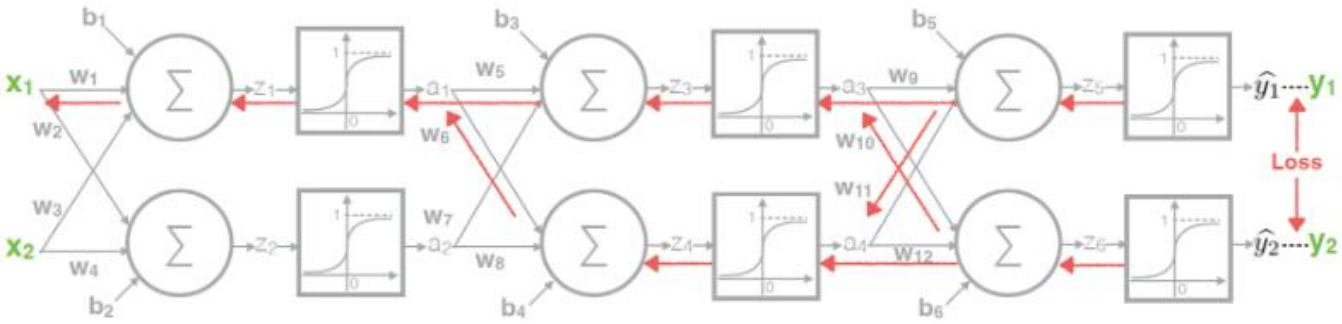
Deep L-layer Neural Networks

- **Forward and Backward propagation**
  - **Forward propagation** refers to the computation and storage of intermediate variables (including outputs) for a neural network in order from the input layer to the output layer.
  - **The back-propagation** runs through the network from the output layer to the input layer storing the intermediate variables (partial derivatives with respect to some parameters) according to the chain rule.



Deep L-layer Neural Networks

- **Loss function - Cost function**
  - The **loss function** quantifies the distance between the actual and predicted value for a single training example.
  - The **cost function averages the losses in the training set** to measure the quality of a model on this data set.



Deep L-layer Neural Networks

- **Loss function - Cost function**

$$\frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i|$$ Mean absolute error (regression)

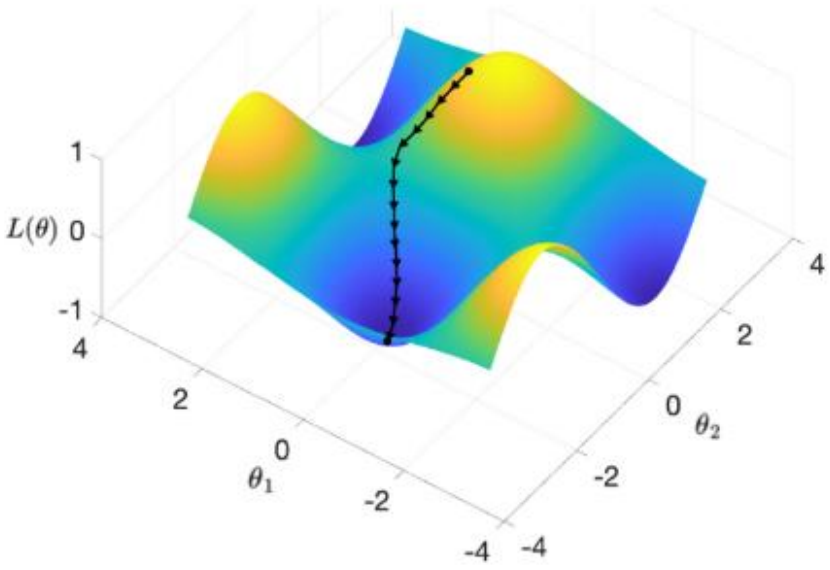
$$\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$ Mean square error (regression)

$$-\frac{1}{N}\sum_{i=1}^{N}(y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i))$$ Cross-entropy (classification)

Deep L-layer Neural Networks

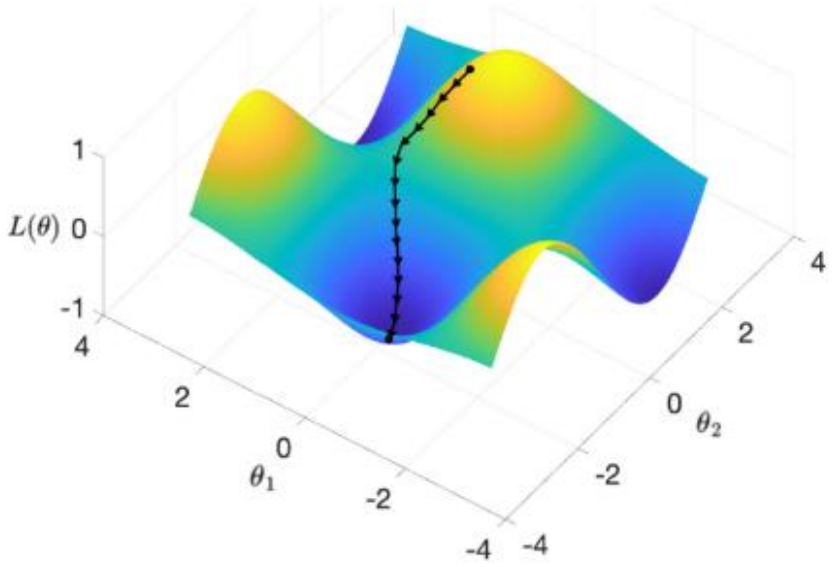
- **Gradient Descent**

  - A **method** to **find** the **minimum of the loss function L** (at least locally) by **iteratively updating** the **parameters** (weights w_i and bias b).
  - The method m**oves over the loss function L** in the direction **marked by the derivatives of L** with **respect to its parameters**.



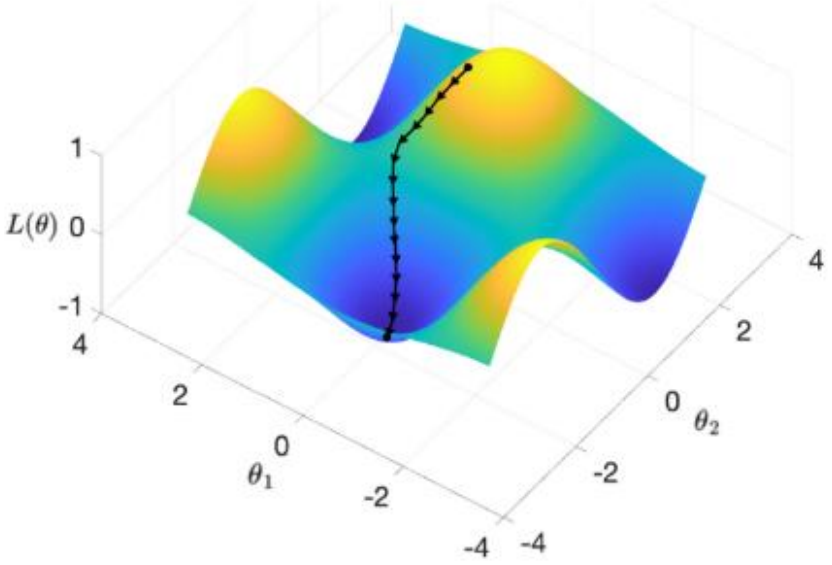
Deep L-layer Neural Networks

- **Gradient Descent**
  - When training neural networks, once the model parameters are initialized, we **alternate forward propagation with back-propagation, updating the model parameters using gradients**.



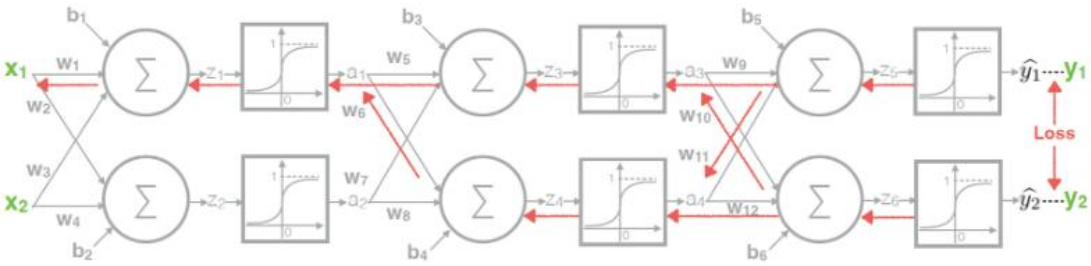
Deep L-layer Neural Networks

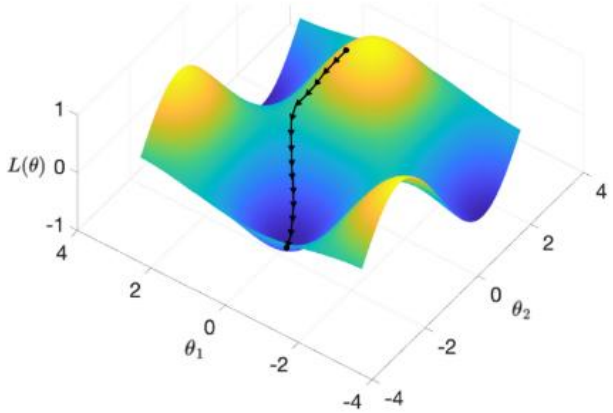
- **Gradient Descent**
  - **Chain rule**
  - The b**ack-propagation** method is mainly **based on** the **"derivative chain rule"** which allows **to calculate** the **partial derivative of the loss function** with **respect** to **each of the parameters (w_i and b)** of each neuron for its correct update.



Deep L-layer Neural Networks

- **Gradient Descent. Derivatives**
  - **Chain rule**





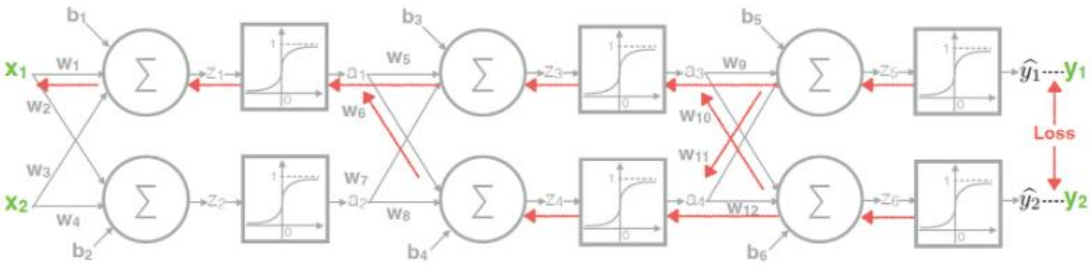
- Loss function $L = \frac{1}{2n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2$
- Parameters $(\theta = \{w_1, w_2, .., w_j, b_1, b_2, .., b_k\})$
- $\theta^{n+1} = \theta^n - \alpha \bigtriangledown L(\theta)$
  - θ^{n+1} is the next position
  - θ^n is the current position
  - α is the learning rate
  - $\bigtriangledown$ the gradient direction of the fastest increase

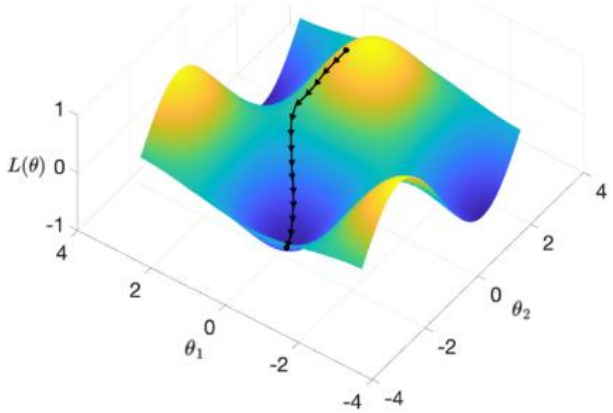
Derivatives

$$\nabla L(\theta) = \left(\frac{\partial L(\theta)}{\partial w_1}, \frac{\partial L(\theta)}{\partial w_2} \dots \frac{\partial L(\theta)}{\partial w_j}, \frac{\partial L(\theta)}{\partial b_1}, \frac{\partial L(\theta)}{\partial b_2} \dots \frac{\partial L(\theta)}{\partial b_k}\right)$$

Deep L-layer Neural Networks

- **Gradient Descent. Learning rate**
  - **Chain rule**





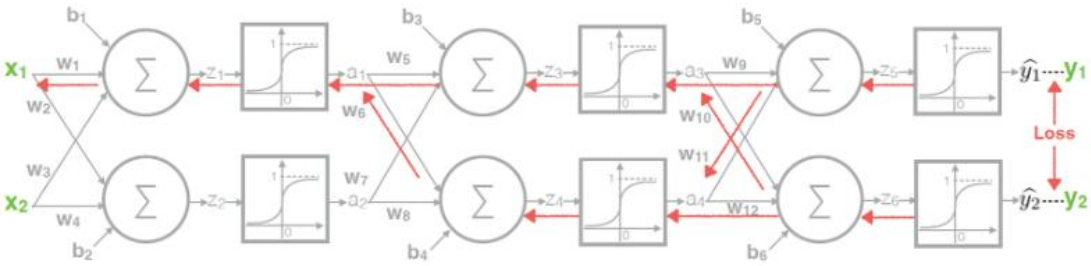
- Loss function $L = \frac{1}{2n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2$
- Parameters $(\theta = \{w_1, w_2, .., w_j, b_1, b_2, .., b_k\})$
- $\theta^{n+1} = \theta^n - \alpha \bigtriangledown L(\theta)$
  - θ^{n+1} is the next position
  - θ^n is the current position
  - α is the learning rate
  - $\bigtriangledown$ the gradient direction of the fastest increase

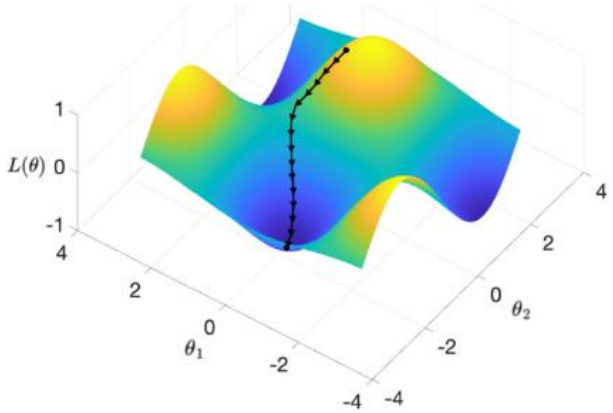
Learning rate

- The **dimension of the steps over the loss function** in the direction of the derivatives to find the minimum is called the learning rate.

Deep L-layer Neural Networks

- **Gradient Descent. Epochs**
  - **Chain rule**





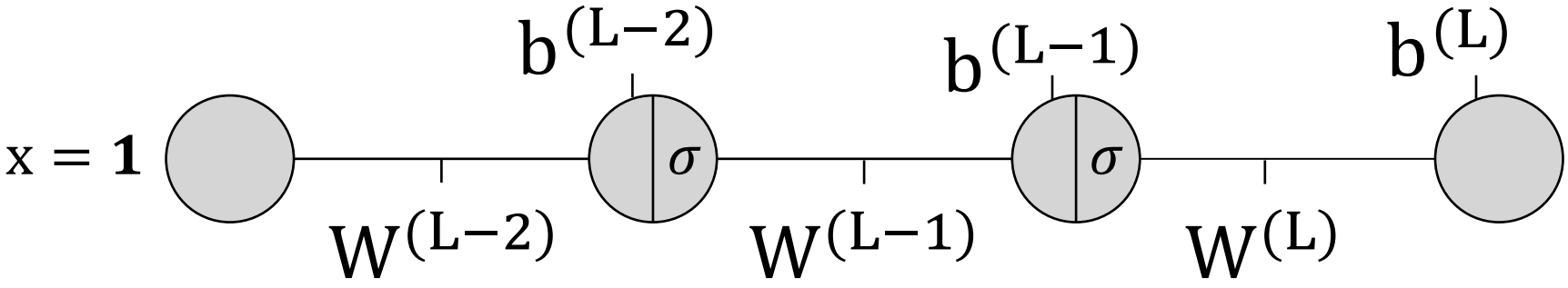
- Loss function $L = \frac{1}{2n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2$
- Parameters $(\theta = \{w_1, w_2, .., w_j, b_1, b_2, .., b_k\})$
- $\theta^{n+1} = \theta^n - \alpha \bigtriangledown L(\theta)$
  - θ^{n+1} is the next position
  - θ^n is the current position
  - α is the learning rate
  - $\bigtriangledown$ the gradient direction of the fastest increase

Epochs

- **Complete iteration of training a machine learning model on a dataset**

Backpropagation Exercise

Exercise 1. Calculate the Forward pass and Backward pass for the proposed Neural Network



$$\boldsymbol{\theta} = \{W^{(L-2)}, b^{(L-2)}, W^{(L-1)}, b^{(L-1)}, W^{(L)}, b^{(L)}\}$$

$$W^{(L-2)} = 0.32 \qquad W^{(L-1)} = 0.18 \qquad W^{(L)} = 0.23$$

$$b^{(L-2)} = 0 \qquad b^{(L-1)} = 0 \qquad b^{(L)} = 0$$

Backpropagation Exercise

Exercise 1. Calculate the Forward pass and Backward pass for the proposed Neural Network

$$L = C_0 = (\hat{y} - y)^2$$ **Cost or loss function**

$$\sigma = \frac{1}{1 + e^{-x}} \qquad \sigma'(x) = \sigma(x)(1 - \sigma(x))$$ **Sigmoid activation function and its derivative**

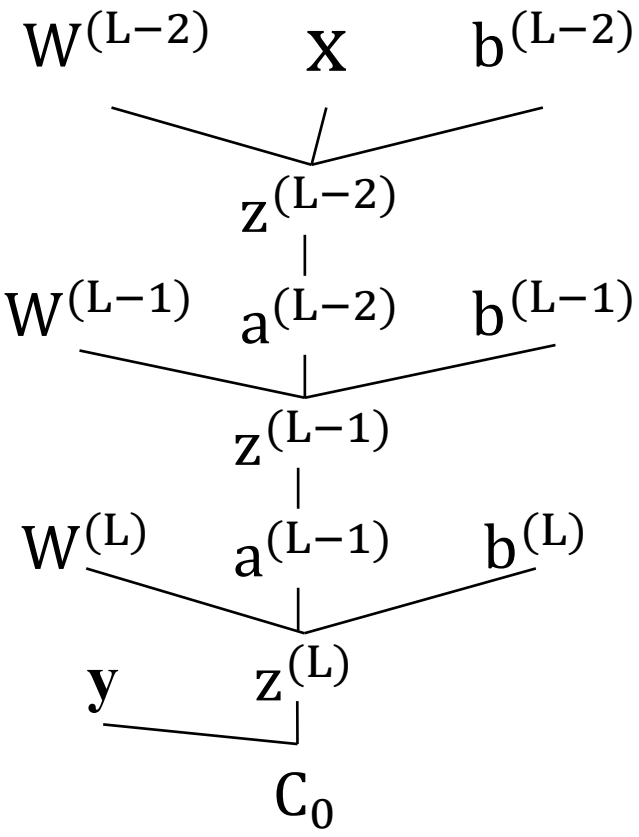
$$z^{(L)} = W^{(L)} a^{(L-1)} + b^{(L)}$$ **Example of neuron input**

$$a^{(L)} = \sigma(z^{(L)})$$ **Neuron activation**

Backpropagation Exercise

Exercise 1. Calculate the Forward pass and Backward pass for the proposed Neural Network

Computation Graph



Gradient of $W^{(L-1)}$

$$\frac{\partial C_0}{\partial W^{(L-1)}} = \frac{\partial z^{(L-1)}}{\partial W^{(L-1)}} \frac{\partial a^{(L-1)}}{\partial z^{(L-1)}} \frac{\partial z^{(L)}}{\partial a^{(L-1)}} \frac{\partial C_0}{\partial z^{(L)}}$$

Backpropagation Exercise

Exercise 1. Calculate the Forward pass and Backward pass for the proposed Neural Network

1. **Obtain outputs (z, a) and the prediction**

2. **Update** $W^{(L)}$ **and** $b^{(L-1)}$, $\alpha = 0.1$