

EL4106 - Inteligencia Computacional

Auxiliar 2

Profesor: Pablo Estévez V.

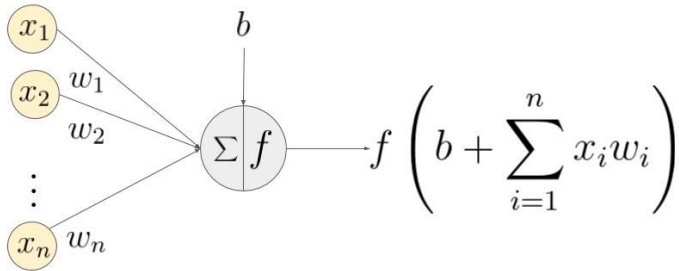
Auxiliar: Pablo Cornejo M.

Ayudantes: Diego Castillo, Andrés González, Sebastián Guzmán, Francisco Soto

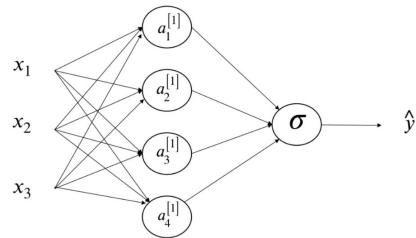
Hoy

1. Tensores
2. Tensores en Colab
3. NN in Pytorch
4. Pytorch en Colab
5. Actividad 2

Representación NN



An example of a neuron showing the input ($x_1 - x_n$), their corresponding weights ($w_1 - w_n$), a bias (b) and the activation function f applied to the weighted sum of the inputs.



$$\begin{aligned}
 z_1^{[1]} &= w_1^{[1]T} x + b_1^{[1]} &\Rightarrow \sigma^{[1]}(z_1^{[1]}) &= a_1^{[1]} \\
 z_2^{[1]} &= w_2^{[1]T} x + b_2^{[1]} &\Rightarrow \sigma^{[1]}(z_2^{[1]}) &= a_2^{[1]} \\
 z_3^{[1]} &= w_3^{[1]T} x + b_3^{[1]} &\Rightarrow \sigma^{[1]}(z_3^{[1]}) &= a_3^{[1]} \\
 z_4^{[1]} &= w_4^{[1]T} x + b_4^{[1]} &\Rightarrow \sigma^{[1]}(z_4^{[1]}) &= a_4^{[1]}
 \end{aligned}$$

→ $a^{[1]}$

$$z_1^{[2]} = w_1^{[2]T} a^{[1]} + b_1^{[2]} \Rightarrow \sigma^{[2]}(z_1^{[2]}) = a_1^{[2]} = \hat{y}$$

Tensores

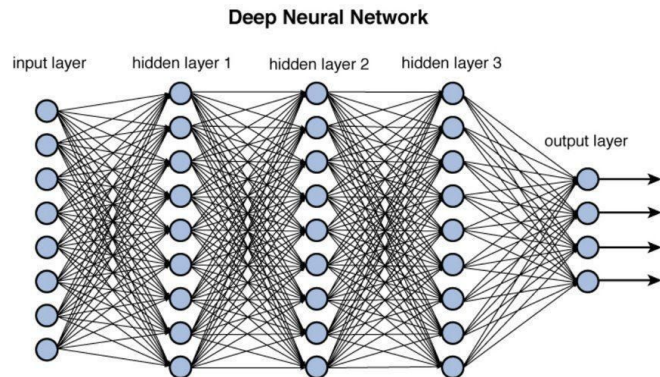
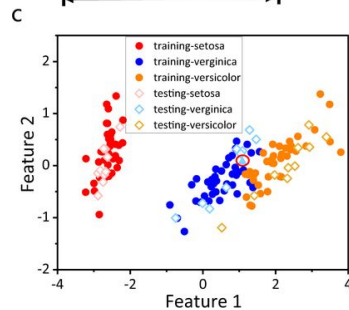
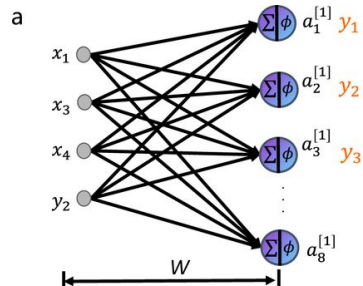


Figure 12.2 Deep network architecture with multiple layers.



Tensores

(Revisar Tensores y operaciones en .ipynb)

- Tensors are the core data structures in PyTorch, similar to multidimensional arrays in NumPy, but optimized for deep learning tasks.
- They can store data in various dimensions (e.g., scalars, vectors, matrices) and support GPU acceleration for high-performance computations.
- Tensors are crucial for defining neural network models, performing operations, and computing gradients during training, making them essential for any PyTorch-based deep learning project.



https://pytorch.org/tutorials/beginner/introyt/tensors_deeper_tutorial.html

Def Red Neuronal Pytorch

- Crear una NN en pytorch

```
import torch
# 'nn' es un paquete dentro de 'torch' que contiene funciones útiles
from torch import nn
# módulo 'functional' del paquete 'nn' contiene funciones útiles
from torch.nn import functional as F
```

Def Red Neuronal Pytorch

- Crear una NN en pytorch

```
class ExampleNN(nn.Module):  
    def __init__(self, in_features, hidden_features, out_features):  
        super(ExampleNN, self).__init__()  
  
        # First layer: Initialize the weight and bias for the first neuron  
        self.W1 = nn.Parameter(torch.randn(hidden_features, in_features))  
        self.b1 = nn.Parameter(torch.randn(hidden_features))  
  
        # Second layer: Initialize the weight and bias for the second neuron  
        self.W2 = nn.Parameter(torch.randn(out_features, hidden_features))  
        self.b2 = nn.Parameter(torch.randn(out_features))  
  
    def forward(self, x):  
        ...
```

Def Red Neuronal Pytorch

- Crear una NN en pytorch

```
class ExampleNN(nn.Module):  
    def __init__(self, in_features, hidden_features, out_features):  
        ...  
  
    def forward(self, x):  
        # Apply first neuron: ReLU( $xW_1^T + b_1$ )  
        x1 = F.relu(F.linear(x, self.W1, self.b1))  
  
        # Apply second neuron: ReLU( $xW_2^T + b_2$ ) using the output of the first neuron  
        x2 = F.linear(x1, self.W2, self.b2)  
  
        return x2  
  
    ...
```


Def Red Neuronal Pytorch

- Crear una NN en pytorch

```
class ExampleNN(nn.Module):  
    def __init__(self, in_features, hidden_features, out_features):  
        ...  
  
    def forward(self, x):  
        # Apply first neuron: ReLU( $xW_1^T + b_1$ )  
        x1 = F.relu(F.linear(x, self.W1, self.b1))  
  
        # Apply second neuron: ReLU( $xW_2^T + b_2$ ) using the output of the first neuron  
        x2 = F.linear(x1, self.W2, self.b2)  
  
        return x2
```

Iris Classification

