Una red neuronal con una neurona

Juan I. Perotti,* Benjamín Marcolongo,† and Martín Abrudsky‡
Instituto de Física Enrique Gaviola (IFEG-CONICET),
Ciudad Universitaria, 5000 Córdoba, Argentina and
Facultad de Matemática, Astronomía, Física y Computación,
Universidad Nacional de Córdoba, Ciudad Universitaria, 5000 Córdoba, Argentina
(Dated: October 14, 2024)

En este trabajo, bla bla bla..XXXXXXXXXX

I. INTRODUCCIÓN

Las redes neuronales... bla bla bla [1].

II. TEORÍA

Bla bla...

III. RESULTADOS

Ver fig. 1.

IV. DISCUSIÓN

La comparación de ... bla bla bla

V. CONCLUSIONES

Concluyendo ...

VI. AGRADECIMIENTOS

JIP, BM y MA agradecen el finaciamiento y el apoyo de CONICET, SeCyT y la UNC.

* juan.perotti@unc.edu.ar
† benjaminmarcolongo@unc.edu.ar
‡ martin.abrudsky@unc.edu.ar

[1] J. A. Hertz, A. S. Krogh, and R. G. Palmer, *Introduction To The Theory Of Neural Computation* (Taylor & Francis Group, Boca Raton London New York, 1999).

Apéndice A: Modelos

1. Modelo 1

El código bla bla bla...

```
## Definimos el primer modelo
class MultiLayerPerceptron(nn.Module):
    def __init__(self):
        super().__init__()

        ## "Achatamos" la matriz de 28x28 píxeles,
        ## transformandola en un vector de 784 elementos
        self.flatten = nn.Flatten()

        ## Definimos el perceptrón multicapa con las
        ## siguientes capas:
        ##
        ## Entrada:        784 neuronas
        ## 1º capa oculta: 512 neuronas
        ## 2º capa oculta: 512 neuronas
```

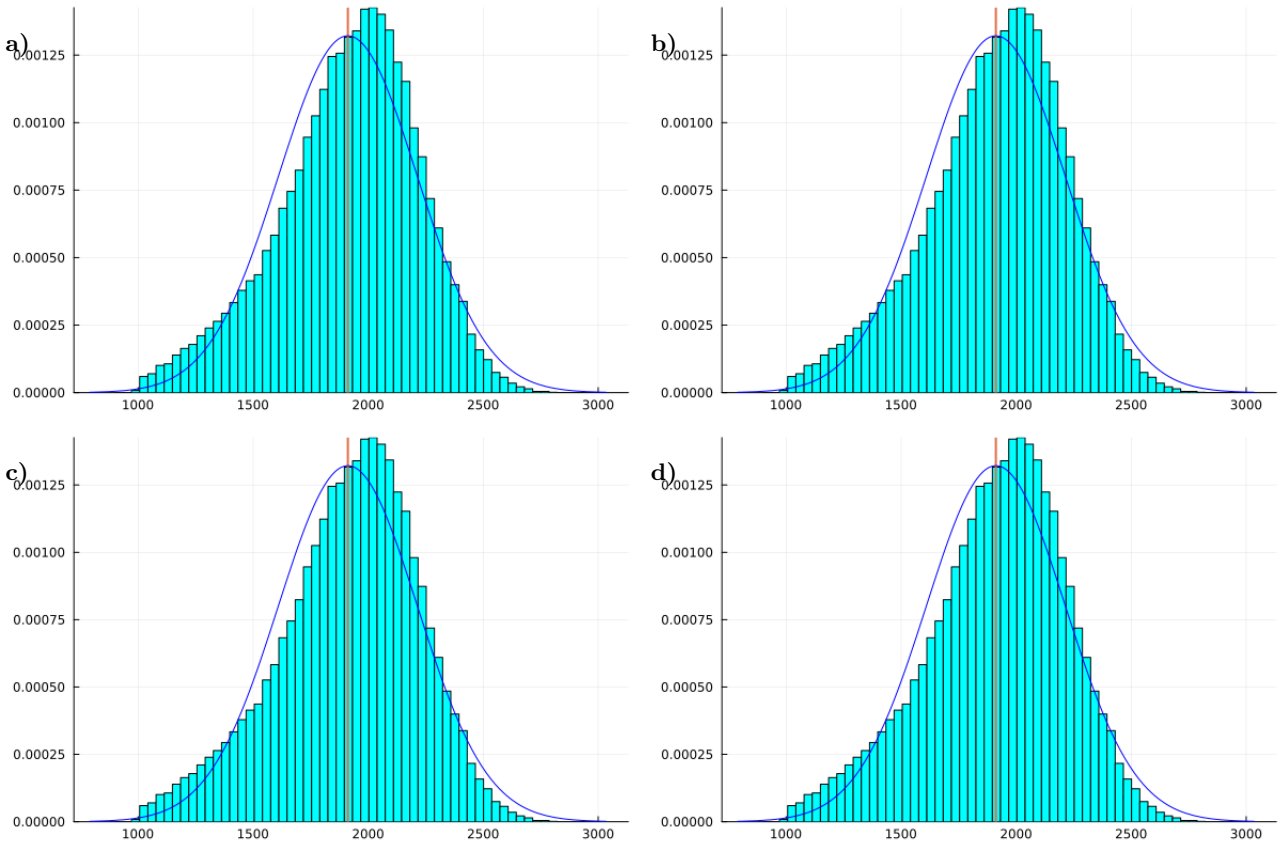

FIG. 1. Un puñado de distribuciones Gaussianas. **a)** Una distribución Gaussiana. **b)** Otra distribución Gaussiana. **c)** Otra distribución Gaussiana más. **d)** Una distribución Gaussiana extra.

```
        ## Salida:         10  neuronas
        ##
        ## Entre capa y capa, utilizamos función de
        ## activación ReLU
        self.linear_relu_stack = nn.Sequential(
            nn.Linear(28*28, 600),
            nn.ReLU(),
            nn.Linear(600, 120),
            nn.ReLU(),
            nn.Linear(120, 10),
            nn.ReLU()
        )

    def forward(self, x):
        x = self.flatten(x)
        x = self.linear_relu_stack(x)
        return x
```

2. Modelo 2

El código bla bla bla...

Apéndice B: Datos

Bla bla bla...