UNIVERSIDAD NACIONAL SANTIAGO ANTÚNEZ DE MAYOLO

FACULTAD DE CIENCIAS

ESCUELA PROFESIONAL INGENIERÍA DE SISTEMAS E INFORMÁTICA

TEMA:

REDES NEURONALES

INTEGRANTES:

- Alvarado Robles Carlos Emilio
- Asis Espinoza Nelson Gustavo
- Blas Obregón Clinton Emilio
- Gonsalez Alva Erika Pilar
- Maldonado Blacido Yanil Alain
- Moreno Trujillo Christian Franco
- Orellano Rondan Carlos Jhardel
- Pagola Pajuelo Angel Wilmer
- Paucar Colonia Frank César
- Silvestre Gutierrez Jefferson Antony

DOCENTE:

Ing° MIGUEL ANGEL SILVA ZAPATA

HUARAZ – PERÚ

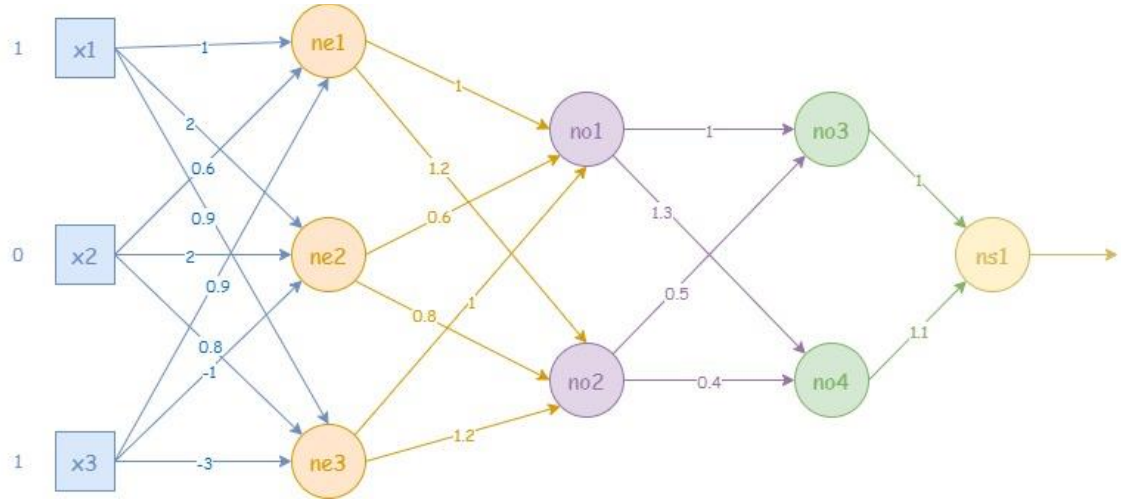
2022

ÍNDICE

I. Enunciado del ejercicio

Dada la siguiente Red Neuronal, implemente su codificación en el lenguaje de programación Python. Siendo su valor de salida deseado igual a 1, su coeficiente de aprendizaje c=1.

II. Diseño de la red neuronal (RN)



III. Proceso de implementación de la red neuronal en Python

1. Primero importamos la librería de la clase “numpy”, lo instanciamos para poder utilizar sus métodos

```
import numpy as np
```

2. Definimos los datos que nos indica el enunciado, el coeficiente de aprendizaje que lo definimos como “alpha = 1”, los valores del vector de entrada que son [x1, x2, x3] donde x1=1, x2=0 y x3=1. También definimos la salida que se solicita que en este caso es 1.

```
alpha = 1  # Coeficiente de aprendizaje
entradas = np.array([[1, 0, 1]])  # Vector de entradas
salida = np.array([[1]])  # Salida deseada: 1
```

3. Continuamos definiendo los pesos que se tienen en cada una de las capas según los datos que se observan en el diseño de la RN.

```
pesos1 = np.array([[1, 2, 0.5], [0.6, 2, 0.8], [0.9, -1, -3]])
pesos2 = np.array([[1, 1.2], [0.6, -0.8], [1, 1.2]])
pesos3 = np.array([[1, 1.3], [0.5, 0.4]])
pesos4 = np.array([[1], [1.1]])
```

4. Seguimos definiendo un contador para obtener el número de interacciones que serán necesarias para que el valor de la salida llegue a ser 1. También definimos una variable para asignarle el error permitido que se pueda tener en el resultado y una variable booleana que nos indique si aún no se ha sobrepasado el error permitido.

```
contador = 0
error_permitido = 0.1
es_mayor_al_error = True
```

5. Luego definimos una función “sigmoide” para aplicarlas en la obtención de las salidas de cada una de las capas. Se solicita un solo parámetro “x” que en este caso viene a ser el vector resultado de la multiplicación entre los valores de entrada con los pesos de las capas.

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

6. Ahora se declara un bucle que se ejecutará mientras el error permitido no se haya sobrepasado. Dentro de ello primero se multiplican las entradas con los pesos de la capa 1 para luego aplicar la función sigmoide con el valor que se obtiene de dicha multiplicación.
   Después se continúa multiplicando los pesos de las siguientes capas con los valores de salida que se obtuvieron de la capa anterior, hasta llegar al valor de la salida final que en este caso es “y4”.

```
while es_mayor_al_error:
    a1 = entradas.dot(pesos1)
    y1 = sigmoide(a1)

    a2 = y1.dot(pesos2)
    y2 = sigmoide(a2)

    a3 = y2.dot(pesos3)
    y3 = sigmoide(a3)

    a4 = y3.dot(pesos4)
    y4 = sigmoide(a4)  # Salida final obtenida
```

7. Se continúa declarando una variable de “error obtenido” que viene a ser la salida que es 1, menos el valor de la salida obtenida “y4”. Se le suma una interacción al contador cada que se realice el proceso del bucle. Y se imprime el numero de interacciones y el error que se obtiene en cada interacción.

```
error_obtenido = salida - y4
contador += 1
print("Iteración ", contador, ":\t 😃 Salida: ",
      y4, " : 💀 Error",  error_obtenido, "\n")
```

8. Se comprueba con una sentencia condicional que el error obtenido no sea mayor al error máximo permitido, si se sobrepasó el error máximo la variable booleana cambia de estado y se termina el proceso de aprendizaje de la RN.

```
if error_obtenido < error_permitido:
    es_mayor_al_error = False
```

9. Y si aún no se sobrepasa el error máximo permitido se realiza el proceso de “Backpropagation” que consiste en propagar los deltas de salida hacia atrás; para cambiar los valores de los pesos de la capa de salida, de las capas intermedias y la capa de entrada sucesivamente. Y volver a realizar todo el proceso del bucle que significaría una nueva interacción de la RN para su proceso de aprendizaje.

```
else:
    # Backpropagation
    delta_4 = y4 * (1.0 - y4) * error_obtenido
    delta_3 = delta_4 * pesos4
    delta_2 = delta_3.T.dot(pesos3)
    delta_1 = delta_2 * pesos2

    pesos4 = pesos4 + alpha * delta_4 * y3.T
    pesos3 = pesos3 + alpha * delta_3 * y2.T
    pesos2 = pesos2 + alpha * delta_2 * y1.T
    pesos1 = pesos1 + alpha * delta_1.dot(y2.T)
```

10. Finalmente, cuando se termine el proceso de aprendizaje se imprime el resultado de la salida final junto al numero de interacciones que fueron necesarias para llegar a la “salida = 1” o su aproximación.

```
print('🏁 La salida final es:', y4 ,'Alcanzado en:', contador, 'interacciones 🏁')
```

IV. Código completo de la RN

```python
import numpy as np

alpha = 1  # Coeficiente de aprendizaje
entradas = np.array([[1, 0, 1]])  # Vector de entradas
salida = np.array([[1]])  # Salida deseada: 1

pesos1 = np.array([[1, 2, 0.5], [0.6, 2, 0.8], [0.9, -1, -3]])
pesos2 = np.array([[1, 1.2], [0.6, -0.8], [1, 1.2]])
pesos3 = np.array([[1, 1.3], [0.5, 0.4]])
pesos4 = np.array([[1], [1.1]])

contador = 0
error_permitido = 0.1
es_mayor_al_error = True


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


while es_mayor_al_error:
    a1 = entradas.dot(pesos1)
    y1 = sigmoide(a1)

    a2 = y1.dot(pesos2)
    y2 = sigmoide(a2)

    a3 = y2.dot(pesos3)
    y3 = sigmoide(a3)

    a4 = y3.dot(pesos4)
    y4 = sigmoide(a4)  # Salida final obtenida

    error_obtenido = salida - y4
    contador += 1
    print("Iteración ", contador, ":\t 😃 Salida: ",
          y4, " : 💀 Error",  error_obtenido, "\n")

    if error_obtenido < error_permitido:
        es_mayor_al_error = False
    else:
        # Backpropagation
        delta_4 = y4 * (1.0 - y4) * error_obtenido
        delta_3 = delta_4 * pesos4
        delta_2 = delta_3.T.dot(pesos3)
        delta_1 = delta_2 * pesos2

        pesos4 = pesos4 + alpha * delta_4 * y3.T
        pesos3 = pesos3 + alpha * delta_3 * y2.T
        pesos2 = pesos2 + alpha * delta_2 * y1.T
        pesos1 = pesos1 + alpha * delta_1.dot(y2.T)

print('🏁 La salida final es:', y4 ,'Alcanzado en:', contador, 'interacciones 🏁')
```

V. Resultados

a) Estos son los resultados cuando el error máximo permitido es igual a 0.1

```
error_permitido = 0.1
```

```
Iteración  3 :  😃 Salida:  [[0.84601981]]  : 💀 Error [[0.15398019]]
Iteración  4 :  😃 Salida:  [[0.85132256]]  : 💀 Error [[0.14867744]]
Iteración  5 :  😃 Salida:  [[0.85628811]]  : 💀 Error [[0.14371189]]
Iteración  6 :  😃 Salida:  [[0.86093725]]  : 💀 Error [[0.13906275]]
Iteración  7 :  😃 Salida:  [[0.86529045]]  : 💀 Error [[0.13470955]]
Iteración  8 :  😃 Salida:  [[0.86936759]]  : 💀 Error [[0.13063241]]
Iteración  9 :  😃 Salida:  [[0.87318779]]  : 💀 Error [[0.12681221]]
Iteración  10 :  😃 Salida:  [[0.87676918]]  : 💀 Error [[0.12323082]]
Iteración  11 :  😃 Salida:  [[0.88012888]]  : 💀 Error [[0.11987112]]
Iteración  12 :  😃 Salida:  [[0.88328289]]  : 💀 Error [[0.11671711]]
Iteración  13 :  😃 Salida:  [[0.8862461]]  : 💀 Error [[0.1137539]]
Iteración  14 :  😃 Salida:  [[0.88903231]]  : 💀 Error [[0.11096769]]
Iteración  15 :  😃 Salida:  [[0.89165431]]  : 💀 Error [[0.10834569]]
Iteración  16 :  😃 Salida:  [[0.89412387]]  : 💀 Error [[0.10587613]]
Iteración  17 :  😃 Salida:  [[0.89645188]]  : 💀 Error [[0.10354812]]
Iteración  18 :  😃 Salida:  [[0.89864839]]  : 💀 Error [[0.10135161]]
Iteración  19 :  😃 Salida:  [[0.90072272]]  : 💀 Error [[0.09927728]]
🏁 La salida final es: [[0.90072272]] Alcanzado en: 19 interacciones 🏁
✓ 0 s  completado a las 0:42
```

Se tiene una salida final [0.90072272] que es un valor muy cercano a 1 y se alcanzó en 19 interacciones.

b) Comprobamos los resultados cuando el valor de error máximo permitido igual a 0.01.

```
error_permitido = 0.01
```

```
Iteración  2440 :  😃 Salida:  [[0.98996812]]  : 💀 Error [[0.01003188]]
Iteración  2441 :  😃 Salida:  [[0.98997015]]  : 💀 Error [[0.01002985]]
Iteración  2442 :  😃 Salida:  [[0.98997217]]  : 💀 Error [[0.01002783]]
Iteración  2443 :  😃 Salida:  [[0.9899742]]  : 💀 Error [[0.0100258]]
Iteración  2444 :  😃 Salida:  [[0.98997622]]  : 💀 Error [[0.01002378]]
Iteración  2445 :  😃 Salida:  [[0.98997824]]  : 💀 Error [[0.01002176]]
Iteración  2446 :  😃 Salida:  [[0.98998026]]  : 💀 Error [[0.01001974]]
Iteración  2447 :  😃 Salida:  [[0.98998228]]  : 💀 Error [[0.01001772]]
Iteración  2448 :  😃 Salida:  [[0.98998429]]  : 💀 Error [[0.01001571]]
Iteración  2449 :  😃 Salida:  [[0.98998631]]  : 💀 Error [[0.01001369]]
Iteración  2450 :  😃 Salida:  [[0.98998832]]  : 💀 Error [[0.01001168]]
Iteración  2451 :  😃 Salida:  [[0.98999034]]  : 💀 Error [[0.01000966]]
Iteración  2452 :  😃 Salida:  [[0.98999235]]  : 💀 Error [[0.01000765]]
Iteración  2453 :  😃 Salida:  [[0.98999436]]  : 💀 Error [[0.01000564]]
Iteración  2454 :  😃 Salida:  [[0.98999637]]  : 💀 Error [[0.01000363]]
Iteración  2455 :  😃 Salida:  [[0.98999838]]  : 💀 Error [[0.01000162]]
Iteración  2456 :  😃 Salida:  [[0.99000039]]  : 💀 Error [[0.00999961]]
🏁 La salida final es: [[0.99000039]] Alcanzado en: 2456 interacciones 🏁
✓ 6 s  completado a las 0:41
```

Se tiene una salida final [0.99000039] que es más cercano a 1 y se alcanzó en 2456 interacciones.

c) Comprobamos los resultados cuando el valor de error máximo permitido igual a 0.001.

```
error_permitido = 0.001
```

```
Iteración  250624 :      😃 Salida:  [[0.99899997]]  : 💀 Error [[0.00100003]]
Iteración  250625 :      😃 Salida:  [[0.99899998]]  : 💀 Error [[0.00100002]]
Iteración  250626 :      😃 Salida:  [[0.99899998]]  : 💀 Error [[0.00100002]]
Iteración  250627 :      😃 Salida:  [[0.99899998]]  : 💀 Error [[0.00100002]]
Iteración  250628 :      😃 Salida:  [[0.99899998]]  : 💀 Error [[0.00100002]]
Iteración  250629 :      😃 Salida:  [[0.99899998]]  : 💀 Error [[0.00100002]]
Iteración  250630 :      😃 Salida:  [[0.99899999]]  : 💀 Error [[0.00100001]]
Iteración  250631 :      😃 Salida:  [[0.99899999]]  : 💀 Error [[0.00100001]]
Iteración  250632 :      😃 Salida:  [[0.99899999]]  : 💀 Error [[0.00100001]]
Iteración  250633 :      😃 Salida:  [[0.99899999]]  : 💀 Error [[0.00100001]]
Iteración  250634 :      😃 Salida:  [[0.99899999]]  : 💀 Error [[0.00100001]]
Iteración  250635 :      😃 Salida:  [[0.999]]  : 💀 Error [[0.001]]
Iteración  250636 :      😃 Salida:  [[0.999]]  : 💀 Error [[0.001]]
Iteración  250637 :      😃 Salida:  [[0.999]]  : 💀 Error [[0.001]]
Iteración  250638 :      😃 Salida:  [[0.999]]  : 💀 Error [[0.001]]
🏁 La salida final es: [[0.999]] Alcanzado en: 250638 interacciones 🏁
✓ 11 min 20 s  completado a las 0:34
```

Se tiene una salida final [0.999] que es más cercano a 1 y se alcanzó en 250638 interacciones.

VI. Conclusión

Si reducimos el error máximo permitido, el valor de la salida será aún más cercano a 1 y la red neuronal demorará mucho más tiempo en el proceso de aprendizaje, es decir requerirá muchas más interacciones.