Universidad de O'Higgins

Tarea N° #2:

# Redes Neuronales

Luis Albandoz González

*COM4402 – Introducción a Inteligencia Artificial*

*Escuela de Ingeniería, Universidad de O'Higgins*

*28 de octubre 2023*

***Resumen*—** El objetivo para esta tarea es utilizar redes neuronales en un problema de clasificación de dígitos. Se utiliza el conjunto de datos *Optical Recognition of Handwritten Digits Data Set* presentando 64 características, con 10 clases y 5620 muestras en total. Las redes neuronales, al ser evaluadas tienen la siguiente estructura: capa de entrada de dimensionalidad 64, una o dos capas ocultas y una capa de salida con 10 neuronas y función de activación softmax.

También se utiliza la función *PyTorch* para entrenar y validar la red neuronal que implementa el clasificador de dígitos, aunque también se implementan las funcionalidades que eviten el sobreajuste (*overfitting*) de la misma.

Con la red neuronal entrenada se implementa una matriz de confusión normalizada donde se evalúa el parámetro *accuracy* señalando cuál red neuronal es más efectiva al evaluar con distintas funciones de activación, tales como ReLu, tanh, sigmoid.

***Palabras claves*—** Redes neuronales, clasificación, perceptrón, funciones de activación, matriz de confusión.

## I. Introducción

Una red neuronal artificial es una rama de la Inteligencia Artificial que consiste en un modelo simplificado que imita la estructura del cerebro humano, está compuesta por una capa de entrada y una capa de salida para una red neuronal simple, y a esto se agrega una o más capas ocultas para una red neuronal multicapa.

La unidad principal de la red neuronal es el perceptrón o neurona que transmite información a las otras neuronas, cada información que transmite el perceptrón es evaluada por la función de activación, y a partir de esta evaluación se dan a conocer las distintas clasificaciones que presentan los datos, estos pueden ser tipo binario y multiclase en caso de que la red neuronal presenta más de dos clases.

## II. Marco Teórico

### A. *Primera Red Neuronal Artificial*

La primera Red Neuronal Artificial fue creada por McCulloch y Pitts en 1943. Según Rodríguez (2021) señala que "fue el primer intento de formalizar matemáticamente el comportamiento de una neurona y de estudiar sus implicaciones en su capacidad de computar y procesar la información" (s.p). A partir de esta afirmación se puede mencionar que el modelo de esta red neuronal fue el primer paso de la evolución de la red neuronal artificial en el campo de Machine Learning

### B. *El primer Perceptrón*

El primer Perceptrón (o perceptrón simple) fue creado por Frank Rosenblatt en 1958 inspirado en mejorar la Red Neuronal de McCulloch y Pitts. Bagnato (2018) indica que:

> Un perceptrón toma varias entradas binarias x1, x2, etc y produce una sola salida binaria. Para calcular la salida, Rosenblatt introduce el concepto de "pesos" w1, w2, etc, un número

> real que expresa la importancia de la respectiva entrada con la salida. (s.p)

Lo anterior quiere decir que cada entrada binaria x1, x2,....., xn está asociada a un peso asociado w1, w2,....., wn. Por lo tanto, se puede asociar a la siguiente sumatoria: ***Sum (wn*xn).***

Según Olabe (1998) señala que el perceptrón "fue originalmente diseñado para el reconocimiento óptico de patrones. Una rejilla de 400 fotocélulas, correspondientes a las neuronas de la retina sensibles a la luz, recibe el estímulo óptico" (p.7). A partir de lo anterior se puede afirmar que las fotocélulas son especies de neuronas que están conectadas a lo largo de la red neuronal y van recogiendo información de otras fotocélulas.

### *C.* ***Perceptrón Multicapa***

El Perceptrón Multicapa surge a partir de la evolución del Perceptrón simple, es decir, compuesta por dos o más neuronas. Según Calvo (2018) menciona que:

> El perceptrón multicapa está compuesto por una capa de entrada, una capa de salida y n capas ocultas entremedias.
> Se caracteriza por tener salidas disjuntas pero relacionadas entre sí, de tal manera que la salida de una neurona es la entrada de la siguiente. (s.p)

A partir de la afirmación anterior se concluye que un perceptrón multicapa está formado por una capa de entrada, al menos formada por una capa oculta y una capa de salida, y las neuronas se encuentran interconectadas.

### *D.* ***Estructuras Neuronales:*** *Feed Forward vs Back Propagation*

Existen 2 tipos de estructuras de una red neuronal artificial que se utilizan con mayor frecuencia: Feed Forward y Back Propagation.

Según Kumar (2022) señala que "cada perceptrón de una capa está vinculado a cada perceptrón de la siguiente capa". A partir de esta afirmación se concluye que los perceptrones de la misma capa no se encuentran conectados. Cada perceptrón conecta a otro perceptrón de la siguiente capa; a partir de esta definición se conoce como retroalimentación hacia adelante o Feed Forward.

Conforme a Johnson (2023) define Back Propagation como:

> El método de ajustar los pesos de una red neuronal en función de la tasa de error obtenida en la época anterior (es decir, iteración). El ajuste adecuado de los pesos permite reducir las tasas de error y hacer que el modelo sea confiable al aumentar su generalización.

Con la afirmación anterior se puede afirmar que al evaluar la red neuronal hacia atrás se pueden ajustar los pesos de las neuronas permitiendo a la red neuronal tener menores errores posibles.

### *E.* ***Funciones de activación***

Hay varias funciones de activación utilizadas en redes neuronales, en este punto se conocerán en profundidad las más utilizadas: Función sigmoide, tangente hiperbólico (Tanh) y rectificador (ReLU).

1) ***Función sigmoide:*** Esta función presenta un comportamiento similar a la función de probabilidad, es decir, los valores de salida están acotados entre 0 y 1.

Según Sotaquirá (2018) señala que "esta función de activación tiene su uso limitado, y realmente su principal aplicación es la clasificación binaria" (s.p) Se puede decir que esta función no tiene un grado de saturación, es decir, si la entrada es un valor mayor a 0 tiende a acercarse al valor de salida 1; por otro lado si el valor de entrada es un valor negativo tiende a acercarse al valor de salida 0, por lo que hay una dificultad en la convergencia de esta función.

2) ***Tangente hiperbólico (Tanh):*** Presenta un comportamiento similar a función sigmoide, pero la diferencia es que los valores de salida están acotados entre -1 y 1.

3) ***Rectificador (ReLU):*** La función ReLU (Rectified Linear Unit) transforma los valores de entrada negativos a 0 y no transforma los valores de entrada positivos manteniéndolos en el mismo valor.

Según Calvo (2020) define que "ReLU es la función más utilizada en el mundo en este momento. Desde entonces, se utiliza en casi todas las redes neuronales convolucionales o en el aprendizaje profundo" (s.p) A partir de esta afirmación se puede señalar que la función de activación más usada es ReLU en el campo de Deep Learning, esto se debe a que el algoritmo de esta función de activación es más simple que las otras funciones.

## III. METODOLOGÍAS

Al entrar en Google Colab se debe cambiar el entorno de ejecución, para ello se debe acceder en el menú superior se debe ir a la pestaña "*Entorno de Ejecución*" haciendo clic en "*cambiar tipo de entorno de ejecución*", y seleccionar/ verificar "GPU" en "*Acelerador de Hardware*".

Posteriormente se utilizan los archivos de texto o datasets: *1_digits_train.txt* y *1_digits_test.txt*, provenientes del repositorio GitHub, por lo que se ejecutan las siguientes líneas de código en Colab.

Al cargar la base de datos se separan en datos de entrenamiento, datos de validación y en datos de prueba, y a la vez se muestran cuántos datos pertenecen a cada grupo mostrado en la (Fig. 1)

```
df_train, df_val = train_test_split(df_train_val, test_size = 0.3, random_state = 1
print("Muestras de entrenamiento: ", len(df_train))
print("Muestras de validación: ", len(df_val))
print("Muestras de prueba: ", len(df_test))
print("Muestras totales: ", len(df_train_val)+len(df_test))
```

```
Muestras de entrenamiento:  3042
Muestras de validación:  1305
Muestras de prueba:  1272
Muestras totales:  5619
```

Fig. 1 Código que muestra la división de los datos de entrenamiento, validación y prueba con los resultados por división.

Posteriormente se crean los *Dataset* y *Dataloaders*, estos últimos que permiten acceder a los elementos del dataset mediante *batches* como se muestra en la Fig. 2

```
# Crear datasets
feats_train = df_train.to_numpy()[:,0:64].astype(np.float32)
labels_train = df_train.to_numpy()[:,64].astype(int)
dataset_train = [ {"features":feats_train[i,:], "labels":labels_train[i]} for i in range(feats_train.shape[0]) ]

feats_val = df_val.to_numpy()[:,0:64].astype(np.float32)
labels_val = df_val.to_numpy()[:,64].astype(int)
dataset_val = [ {"features":feats_val[i,:], "labels":labels_val[i]} for i in range(feats_val.shape[0]) ]

feats_test = df_test.to_numpy()[:,0:64].astype(np.float32)
labels_test = df_test.to_numpy()[:,64].astype(int)
dataset_test = [ {"features":feats_test[i,:], "labels":labels_test[i]} for i in range(feats_test.shape[0]) ]

# Crear dataloaders
dataloader_train = torch.utils.data.DataLoader(dataset_train, batch_size=128, shuffle=True, num_workers=0)
dataloader_val = torch.utils.data.DataLoader(dataset_val, batch_size=128, shuffle=True, num_workers=0)
dataloader_test = torch.utils.data.DataLoader(dataset_test, batch_size=128, shuffle=True, num_workers=0)
```

(Fig. 2) Código que muestra la creación de dataset y dataloaders.

En el siguiente pseudocódigo que se muestra a continuación se entrena el modelo por 1000 épocas, procesando cada batch o lote en que los dataloaders particionar los datos como se muestra en la (Fig. 3)

```
# Guardar resultados del loss y epocas que duró el entrenamiento
loss_train = []
loss_val = []
epochs = []

# Entrenamiento de la red por n epocas
for epoch in range(1000):

  # Guardar loss de cada batch
  loss_train_batches = []
  loss_val_batches = []

  # Entrenamiento ----------------------------------------------------------------
  model.train()
  # Debemos recorrer cada batch (lote de los datos)
  for i, data in enumerate(dataloader_train, 0):
    # Procesar batch actual
    inputs = data["features"].to(device) # Características
    labels = data["labels"].to(device)   # Clases
    # zero the parameter gradients
    optimizer.zero_grad()
```

(Fig. 3) Pseudocódigo que muestra el entrenamiento por 1000 épocas.

Se evalúan los siguientes modelos de redes neuronales por 1000 épocas, por lo cual se deben modificar algunos parámetros:

- 10 neuronas en la capa oculta, usando función de activación ReLU como se muestra en la (Fig. 4)

```
model = nn.Sequential(
          nn.Linear(64, 10),
          nn.ReLU(),
          nn.Linear(10,10)
        )
```

(Fig. 4) Modelo de red neuronal con 10 neuronas en capa oculta usando ReLU.

- 40 neuronas en la capa oculta, usando función de activación ReLU mostrándose en la Fig. 5

```
model = nn.Sequential(
          nn.Linear(64, 40),
          nn.ReLU(),
          nn.Linear(40,10)
        )
```

(Fig. 5) Modelo de red neuronal con 40 neuronas en capa oculta usando ReLU.

- 10 neuronas en la capa oculta, usando función de activación Tanh como se muestra en la Fig. 6

```
model = nn.Sequential(
          nn.Linear(64, 10),
          nn.Tanh(),
          nn.Linear(10,10)
        )
```

(Fig. 6) Modelo de red neuronal con 10 neuronas en capa oculta usando Tanh.

- 40 neuronas en la capa oculta, usando función de activación Tanh en donde se muestra en la (Fig. 7)

```
model = nn.Sequential(
          nn.Linear(64, 40),
          nn.Tanh(),
          nn.Linear(40,10)
        )
```

(Fig. 7) Modelo de red neuronal con 40 neuronas en capa oculta usando Tanh.

**Importante**: Se debe considerar que antes de entrenar las redes neuronales mencionadas se debe dejar como comentario el siguiente cuadro de código que se entrena a una red neuronal con 2 capas ocultas en el que se muestra en la Fig. 8

```
# Este código se utiliza exlusivamente para los casos 5 y 6:
#NOTA: El código anterior a este se debe dejar como comentario utilizando # por línea de cód

#-- Modelo de una capa oculta con 10 neuronas y activación ReLU --
# Capa de entrada de 64 neuronas (porque hay 64 caracteristicas)
# 2 capas ocultas de 10 neuronas con activación ReLU
# Capa de salida de 10 neuronas (porque hay 10 clases)

#model = nn.Sequential(
#          nn.Linear(64, 10),
#          nn.ReLU(),
#          nn.Linear(10,40),
#          nn.ReLU(),
#          nn.Linear(40,10)
#        )
```

(Fig. 8) Línea de código del modelo de red neuronal con dos capas ocultas

Antes de entrenar las neuronas que presentan 2 capas ocultas, en la línea de código que se muestran desde la Fig. 4 hasta la Fig. 7 se debe dejar expresado en comentario como se muestra en la Fig. 9

```
#model = nn.Sequential(
#          nn.Linear(64, 40),
#          nn.Tanh(),
#          nn.Linear(40,10)
#        )
```

(Fig. 9) Línea de código del modelo de red neuronal con una capa oculta

- 2 capas ocultas con 10 y 10 neuronas cada una y función de activación ReLU como se muestra en la Fig. 10

```
model = nn.Sequential(
          nn.Linear(64, 10),
          nn.ReLU(),
          nn.Linear(10,10),
          nn.ReLU(),
          nn.Linear(10,10)
        )
```

(Fig. 10) Línea de código del modelo de red neuronal con dos capas ocultas con 10 y 10 neuronas utilizando ReLU.

- 2 capas ocultas con 40 y 40 neuronas cada una y función de activación ReLU en el que se ilustra en la Fig. 11

```
model = nn.Sequential(
          nn.Linear(64, 10),
          nn.ReLU(),
          nn.Linear(10,40),
          nn.ReLU(),
          nn.Linear(40,10)
        )
```

Fig. 11 Línea de código del modelo de red neuronal con dos capas ocultas con 40 y 40 neuronas utilizando ReLU.

Observando el algoritmo para ver la matriz de confusión y el valor de *accuracy*. Para ello se analizarán los datos de entrenamiento y de validación normalizados, por lo cual se deben instalar algunas librerías en Colab como se muestra en la (Fig. 12)

```
from sklearn.metrics import confusion_matrix, accuracy_score
import matplotlib.pyplot as plt
import seaborn as sns
```

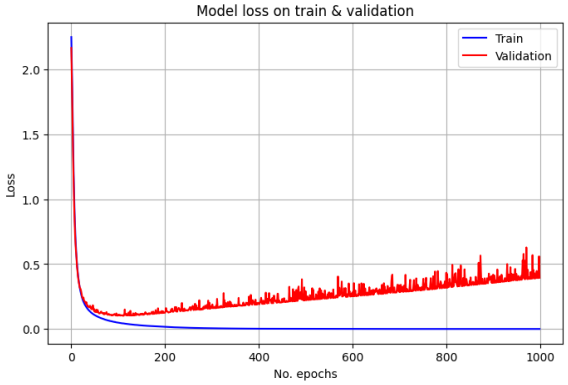
Fig. 12 Línea de código que muestran la instalación de librerías.

## IV. ANÁLISIS Y RESULTADOS OBTENIDOS

Al entrenar cada red neuronal artificial se pueden obtener el tiempo *loss* de entrenamiento, de validación, el gráfico *loss* de entrenamiento y de validación. El valor de *accuracy* y la matriz de confusión para los conjuntos de entrenamiento para las siguientes redes neuronales:

- **10 neuronas en la capa oculta, usando función de activación ReLU:**

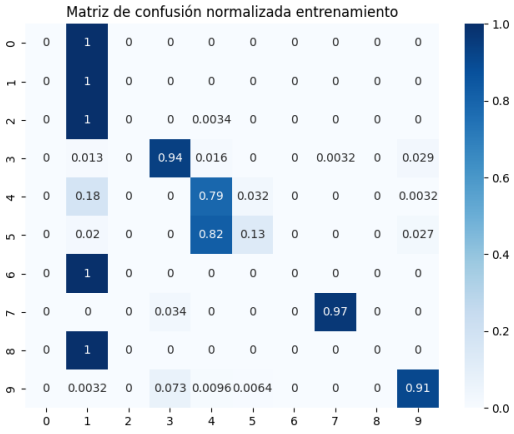
Gráfico *loss* entrenamiento y validación:




(Fig. 13) Gráfico loss entrenamiento y validación

***Accuracy entrenamiento:*** *0.4806048652202498*

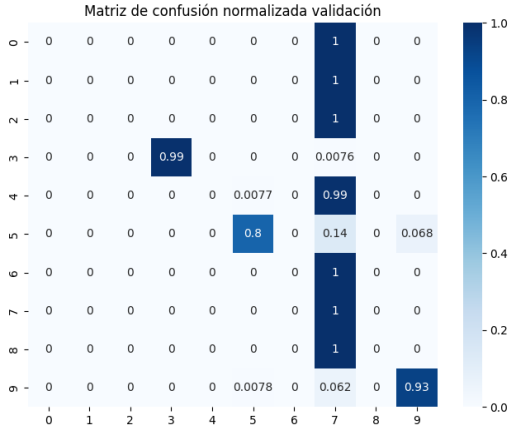
Matriz de confusión entrenamiento:




(Fig. 14) Matriz de confusión entrenamiento

***Accuracy validación:*** *0.3831417624521073*

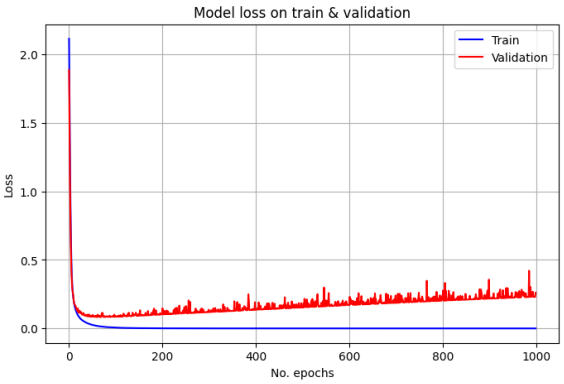
Matriz de confusión validación:




(Fig. 15) Matriz de confusión validación

- **40 neuronas en la capa oculta, usando función de activación ReLU:**

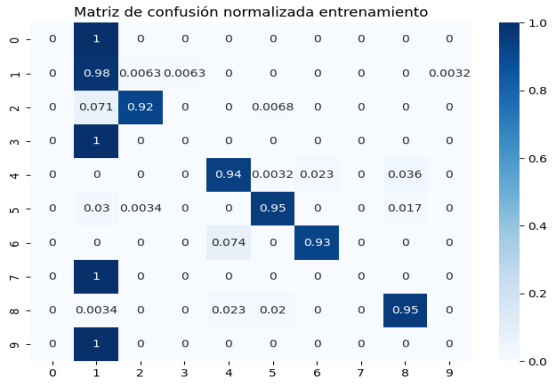
Gráfico *loss* entrenamiento y validación:




(Fig. 16) Gráfico loss entrenamiento y validación

***Accuracy entrenamiento:*** *0.5670611439842209*

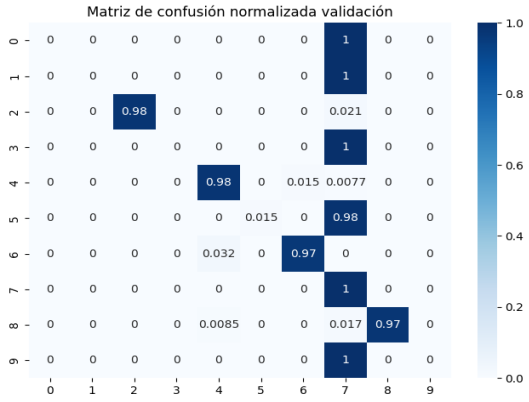
Matriz de confusión entrenamiento:




(Fig. 17) Matriz de confusión entrenamiento

***Accuracy validación:*** *0.496551724137931*

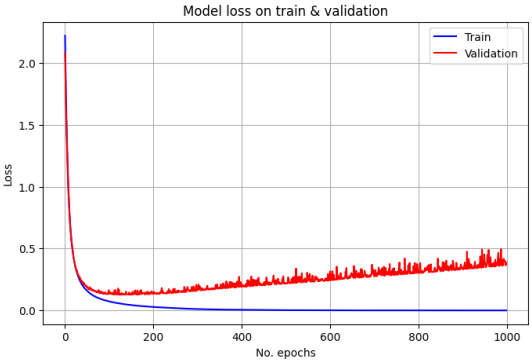
Matriz de confusión validación:




(Fig. 18) Matriz de confusión validación

- **10 neuronas en la capa oculta, usando función de activación Tanh:**

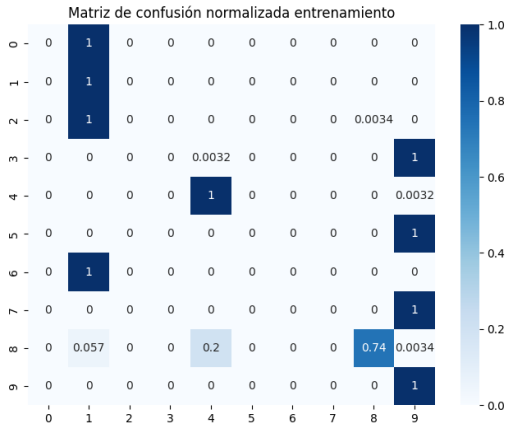
Gráfico *loss* entrenamiento y validación




(Fig. 19) Gráfico loss entrenamiento y validación

***Accuracy entrenamiento:*** *0.3809993425378041*

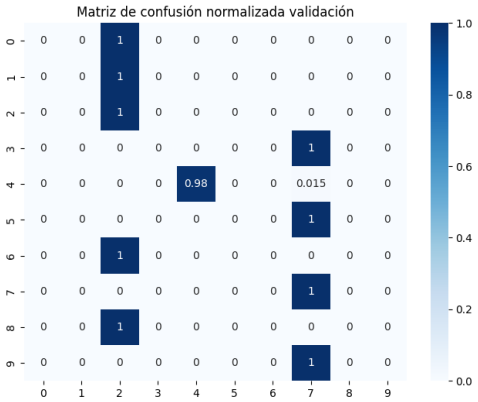
Matriz de confusión entrenamiento:




(Fig. 20) Matriz de confusión entrenamiento

***Accuracy validación:*** *0.31877394636015327*

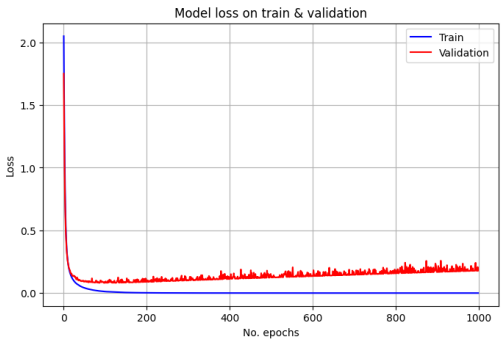
Matriz de confusión validación:




(Fig. 21) Matriz de confusión validación

- **40 neuronas en la capa oculta, usando función de activación Tanh:**

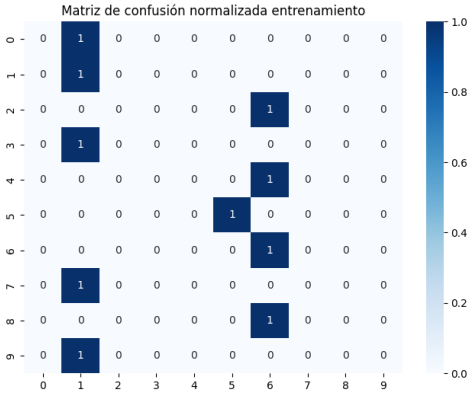
Gráfico loss entrenamiento y validación:




(Fig. 22) Gráfico loss entrenamiento y validación

***Accuracy entrenamiento:*** *0.3034188034188034*

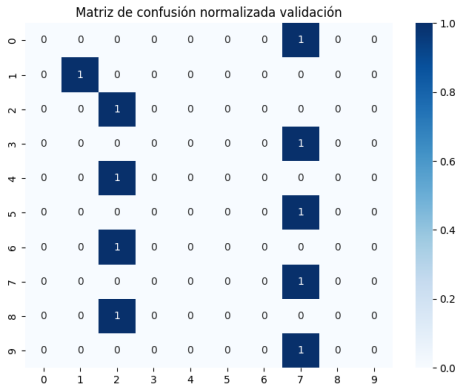
Matriz de confusión entrenamiento:



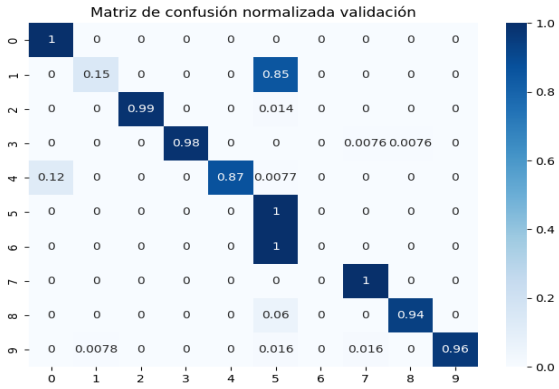

(Fig. 23) Matriz de confusión entrenamiento

***Accuracy validación:*** *0.3157088122605364*

Matriz de confusión validación:



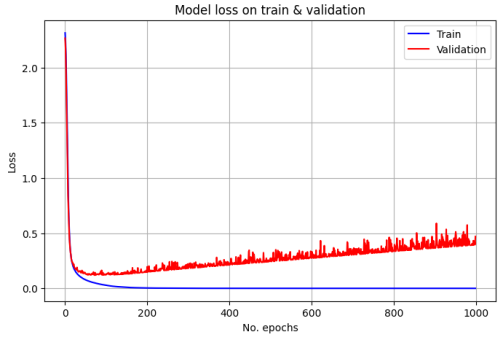

(Fig. 24) Matriz de confusión validación




(Fig. 27) Matriz de confusión validación.

- **2 capas ocultas con 10 y 10 neuronas cada una y función de activación ReLU:**

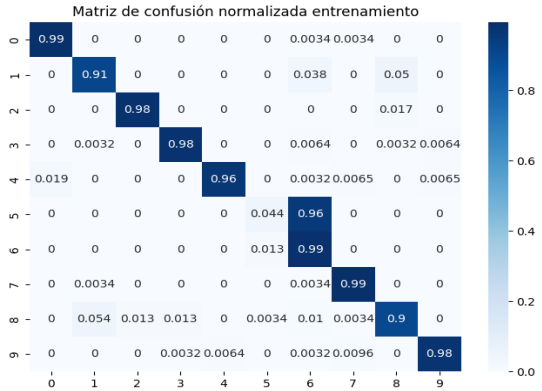
Gráfico loss entrenamiento y validación:




(Fig. 25) Gráfico loss entrenamiento y validación.

***Accuracy entrenamiento:*** *0.8757396449704142*

Matriz de confusión entrenamiento:



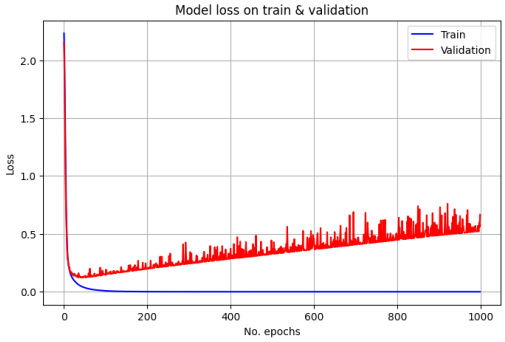

(Fig. 26) Matriz de confusión entrenamiento.

***Accuracy validación:*** *0.7969348659003831*

Matriz de confusión validación:

- **2 capas ocultas con 40 y 40 neuronas cada una y función de activación ReLU:**

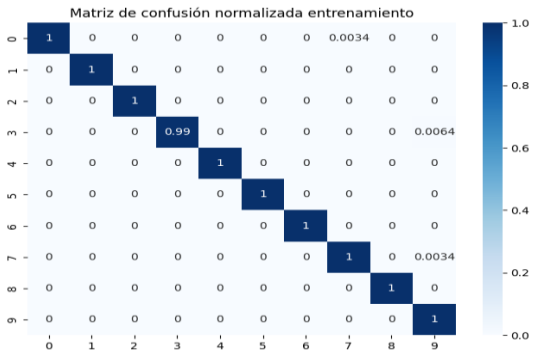
Gráfico loss entrenamiento y validación:




(Fig. 28) Gráfico loss entrenamiento y validación.

***Accuracy entrenamiento:*** *0.9986850756081526*

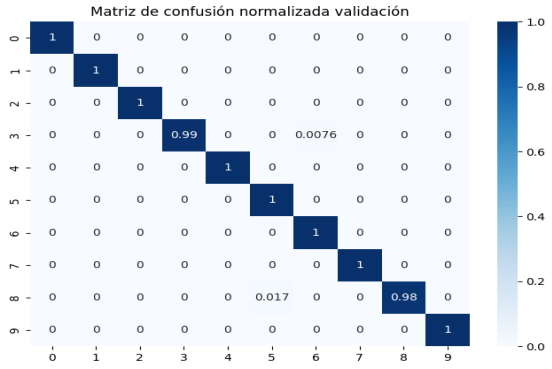
Matriz de confusión entrenamiento:




(Fig. 29) Matriz de confusión entrenamiento.

***Accuracy validación:*** *0.9977011494252873*

Matriz de confusión validación:




(Fig. 30) Matriz de confusión validación.

Al variar la cantidad de neuronas en una capa oculta de una red neuronal (aumento de 10 neuronas a 40 neuronas) si se utiliza la función de activación ReLU se ve un aumento en el accuracy o en la exactitud en que la red neuronal clasifica los datos, en el conjunto de entrenamiento es levemente mayor accuracy que el conjunto de validación.

Si al aumentar las capas ocultas en la red neuronal se puede ver un aumento considerado en la medida de la exactitud (o accuracy) en la red neuronal como se logra ver al utilizar la función de activación ReLU.

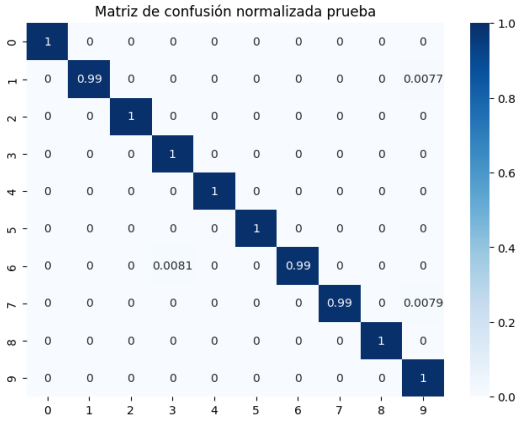
El efecto de las funciones de activación resulta ser variado. En la función ReLU al aumentar las neuronas en la capa oculta se experimenta un aumento del accuracy, sin embargo, cuando hay más capas ocultas utilizando la función ReLU hay un aumento muy significativo en la exactitud de clasificar los datos.
Pero en la función Tanh al aumentar las neuronas en una capa oculta se observa la disminución en la exactitud de la clasificación de los datos, tanto en el conjunto de entrenamiento como en el conjunto de validación.

Al observar las matrices de confusión de las redes neuronales evaluadas se pueden deducir que al obtener mayor exactitud (o accuracy) es proporcional a obtener mayor caso de verdadero positivo, es decir, por cada valor predecido a una clase corresponde a su clase real, esto es, si se observa en el primer cuadro superior del lado izquierdo se pueden apreciar el valor 1 (porcentual) o un número porcentual muy cercano a 1, como se observan en las dos últimas redes neuronales.

Por otra parte, si se analizan los accuracies de los modelos de las redes neuronales en el conjunto de validación se puede concluir que la última red neuronal (2 capas ocultas con 40 y 40 neuronas cada una y función de activación ReLU) presenta mayor accuracy de aproximadamente 0.99; por lo que se analizará la matriz de confusión y el accuracy en el conjunto de prueba para esta red neuronal:

***Accuracy prueba:*** *0.9976415094339622*

Matriz de confusión prueba:




(Fig. 31) Matriz de confusión prueba.

Analizando la matriz de confusión y el accuracy obtenido en el conjunto de prueba para la red neuronal seleccionada si se compara con el conjunto de validación se puede ver el valor del accuracy muy similar, debido a que el accuracy del conjunto de validación tiene una exactitud milimétricamente mejor (0.9977 vs 0.9976). Si se analizan las matrices de confusión de la Fig. 30 y Fig. 31 se pueden observar una similitud en las diagonales que presentan valores porcentuales iguales o muy cercanos a 1, a partir de esta observación se puede deducir que hay un gran porcentaje de verdaderos positivos para el conjunto de validación y el conjunto de prueba.

## V. CONCLUSIONES GENERALES

Las redes neuronales es una de las técnicas más utilizadas en el campo de machine learning que consiste principalmente en imitar la estructura del cerebro humano. La parte más importante de la red neuronal es el perceptrón que consiste en transmitir

información a otras neuronas que se encuentran interconectadas.

Las funciones de activación más usadas en las redes neuronales son la función ReLU, tangente hiperbólico (Tanh) y sigmoide. La función ReLU elimina valores negativos de entrada, por lo que los transforma a 0, pero mantiene los valores de entrada si son positivos; en cambio, los valores de salida que resultan en el tangente hiperbólico varían entre -1 y 1; por su parte la función sigmoide presenta un comportamiento similar al tangente hiperbólico (Tanh), pero la diferencia en que los valores de salida varían entre 0 y 1 (muy parecido a la función de probabilidad).

Al analizar las redes neuronales se pueden deducir que la función que presenta mayor exactitud en los datos (accuracy) es la función ReLU, esto es debido a una mayor exactitud en la clasificación que presentan los datos, uno de los factores que se explican es una mayor cantidad de neuronas que hay en la capa oculta, como se observan en las evaluaciones del modelo 1 y modelo 2. También si se aumentan las capas ocultas en las redes neuronales y utilizando la función ReLU, al analizar el modelo 5 y modelo 6 de la red neuronal se puede evidenciar que el modelo 6 tiene una mayor efectividad en la exactitud de clasificar datos, tanto en el grupo de entrenamiento como en el grupo de validación presentando un accuracy cercano al 1 (en valores porcentuales).

Por otra parte, al analizar el modelo 3 y modelo 4 de la red neuronal se utiliza la función de activación Tanh al medir los datos no resultan ser exactos comparando con la función ReLU, en el modelo 3 presenta un accuracy mayor en el conjunto de entrenamiento comparando con el conjunto 4 (0.38 vs 0.30) esto se debe a que en el modelo 4 presenta mayor cantidad de neuronas en la capa oculta, por lo que conviene tener menos neuronas en la capa oculta para tener una mayor exactitud en la clasificación de los datos si se utiliza la función Tanh.

La matriz de confusión resulta ser fundamental en cómo se visualiza el desempeño de una red neuronal, a mayor exactitud o accuracy es proporcional a tener un mayor valor porcentual en la medición en los verdaderos positivos, esto es, al predecir la clase, verdaderamente corresponde a esta clase. En particular el modelo 6 presenta un mejor desempeño y una mayor cantidad de verdaderos positivos, ya sea en el conjunto de entrenamiento, validación y prueba.

## REFERENCIAS BIBLIOGRÁFICAS


[1] Rodríguez, R. (2021). Neuronas de McCulloch y Pitts. La Máquina Oráculo. https://lamaquinaoraculo.com/deep-learning/el-modelo-neuronal-de-mcculloch-y-pitts/

[2] Bagnato, J. (2018). Breve Historia de las Redes Neuronales Artificiales. Aprende Machine Learning en Español. https://www.aprendemachinelearning.com/breve-historia-de-las-redes-neuronales-artificiales/

[3] Olabe, X. B. (1998). Redes neuronales artificiales y sus aplicaciones. Publicaciones de la Escuela de Ingenieros. https://ocw.ehu.eus/pluginfile.php/40137/mod_resource/content/1/redes_neuro/contenidos/pdf/libro-del-curso.pdf

[4] Calvo, D. (2018). Perceptrón Multicapa - Red Neuronal. Diego Calvo. https://www.diegocalvo.es/perceptron-multicapa/

[5] Kumar, S. (2022). What is Feed-Forward Concept in Machine Learning?. Aitude. https://www.aitude.com/what-is-feed-forward-concept-in-machine-learning/

[6] Johnson, D. (2023). Back Propagation in Neural Network: Machine Learning Algorithm. GURU99. https://www.guru99.com/backpropogation-neural-network.html

[7] Sotaquirá, M. (2018). La Función de Activación. codificandobits. https://www.codificandobits.com/blog/funcion-de-activacion/

[8] Calvo, J. (2020). Crear Red Neuronal desde las matemáticas. European Valley. https://www.europeanvalley.es/noticias/crear-red-neuronal-desde-las-matematicas