

Project 2: Implementation of a deep neural network model for regression or classification

Universidad de Monterrey

School of Engineering and Technologies

Virgilio Del Bosque Luna 578255 | Ingeniería en Mecatrónica

Víctor Manuel Contreras González 625627 | Ingeniería en Mecatrónica

Marcelo Garza Rodríguez 583252 | Ingeniería en Gestión Empresarial

Course: Artificial Intelligence II

Lecturer: Dr. Andrés Hernández Gutiérrez

Due date: Saturday, 19 October 2023, San Pedro Garza García, Nuevo León

"Damos nuestra palabra de que hemos realizado esta actividad con integridad académica"

Learning objective

This project aims to provide students with a hands-on understanding of deep neural networks and their application in classification or regression tasks using TensorFlow. When completing this project, each team will: a) fully understand the basic structure and functionality of a deep neural network, b) implement and apply a deep neural network model to a classification OR linear regression task, c) improve the deep neural network model using Batch Normalisation and/or Dropout, and d) analyse the network's performance and understand its limitations.

Project Outline

This project consists of both theoretical exploration and practical implementation of a perceptron model using TensorFlow.

Índice

1. [Deep Neural Network Model](#)
2. [Activation Functions](#)
3. [Use Case Application](#)
4. [Practical Implementation using TensorFlow in Python](#)
 1. [Dataset](#)
 2. [Exploratory Data Analysis](#)
 3. [Data Preparation](#)
 4. [Model Design and Development](#)
 - [Design your perceptron model using TensorFlow](#)
 - [Train your model](#)
 - [Assess the training process using the learning curves](#)
 - [Perform predictions using the trained model](#)
5. [Discussions and Conclusions](#)
6. [Referencias](#)

Theoretical Research

1. Deep neural network model

Una red neuronal con una sola neurona se llama perceptrón y puede dividirse en dos tipos. El primero es el perceptrón de una sola capa, que solo puede aprender patrones linealmente separables. El segundo es el perceptrón multicapa, que tiene dos o más capas, lo que le da un mayor poder de procesamiento y permite aprender patrones más complejos (Banoula, 2024).

El perceptrón es la red neuronal más sencilla, diseñada para procesar una entrada y generar una salida. Funciona a través de varios elementos: las entradas, que son los datos iniciales; los pesos, que asignan importancia a cada entrada; y los sesgos, que ajustan el resultado final. Luego,

la función de activación procesa las entradas y los sesgos para producir la salida, que representa la predicción final del perceptrón.

```
from IPython.display import Image, display
imagen = '/content/Perceptron.png'
display(Image(filename=imagen, width=200, height=300))
```



Figura 1. Estructura básica de un perceptrón

✓ Fórmula del Perceptrón

La salida del perceptrón se define como:

$$f(x) = \begin{cases} 1 & \text{si } x \cdot w + b \geq 0 \\ 0 & \text{en otro caso} \end{cases}$$

donde:

$(x \cdot w)$ Es el producto escalar de las entradas y los pesos.

(b) Es el bias.

$f(x)$ Función de activación escalón, que devuelve 1 si la suma ponderada es mayor o igual a 0, y 0 en caso contrario.

Feedforward Neural Network

Feedforward Neural networks (FFNs), son aquellas redes neuronales que se componen tienen dentro de su estructura 1 o 2 capas ocultas, a diferencia de las redes neuronales profundas que pueden tener un número n de capas ocultas. (McGonagle et al., s. f.)

```
from IPython.display import Image, display
imagen = '/content/FNNarq.png'
display(Image(filename=imagen, width=400, height=200))
```

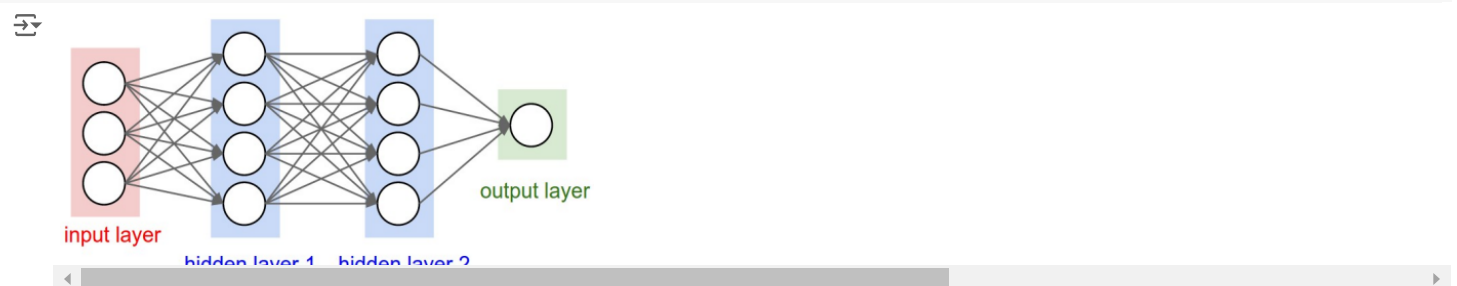


Figura 2. Arquitectura de una FNN

✓ Ventajas y Limitaciones

Las FNN tienen la ventaja de que aprenden a partir de ejemplos sin necesitar un algoritmo específico de solución, similar al aprendizaje humano, y poseen una capacidad de generalización que les permite reconocer patrones nuevos similares a los de entrenamiento. Sin embargo, su rendimiento no está garantizado para todos los problemas, ya que depende de la correcta selección de parámetros como el número de capas ocultas y el algoritmo de entrenamiento. Además, no existe una metodología clara para definir la arquitectura óptima, lo que implica un

proceso de prueba y error repetitivo que consume tiempo. Finalmente, estas redes pueden quedarse atrapadas en mínimos locales durante el entrenamiento, lo que afecta su desempeño en problemas complejos. (Benardos & Vosniakos, 2006)

Capas Ocultas

De acuerdo con Haykin (1999), la función de una capa oculta es intervenir entre la información que la red neuronal recibe y la información de salida de la red neuronal, de una manera útil, está explicación, aunque bastante vaga, nos sienta la premisa de que las capas ocultas recibirán la información, realizarán algún proceso con ella, y después de transformarla, la entregarán a la salida, pero entonces ¿Qué es este proceso que realiza la capa oculta?

Las capas ocultas, agregan pesos a los inputs que está recibiendo la red, y los pasan por una función de activación, de manera que pueden encontrar relaciones no lineales entre los inputs y outputs de los datos. (DeepAI, 2020)

Este proceso se lleva a cabo de la siguiente manera:

Suponiendo que tenemos dos entradas, y la primera capa oculta cuenta con 3 neuronas, cada una de estas neuronas estará conectada con ambas entradas, y básicamente lo que se realiza para cada neurona de la capa oculta es, sumar la primera entrada multiplicada por un peso a la segunda entrada multiplicada por un peso, y agregarle un sesgo:

$$Z_1^{(1)} = w_{11}^{(1)} x_1 + w_{12}^{(1)} x_2 + b_1^{(1)}$$

Donde, w son los pesos, x es la entrada y b son los bias.

El resultado de esto se guarda en $a_1^{(1)} = \sigma(z_1^{(1)})$, de esta manera, entonces, al momento de que la información viaja a la siguiente capa oculta, no es x (entrada) quien llega a la próxima neurona, sino $a^{(1)}$ (en este caso), de manera que entonces la ecuación se vería así:

$$Z_1^{(2)} = w_{11}^{(2)} a_1^{(1)} + w_{12}^{(2)} a_2^{(1)} + b_1^{(2)}$$

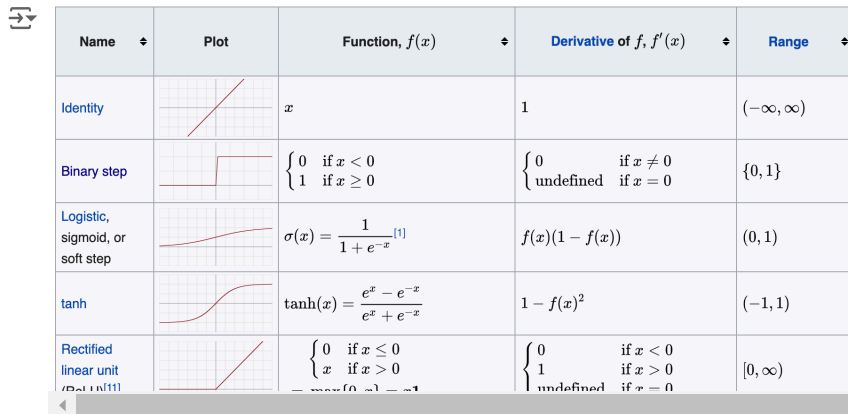
Y este proceso sigue hasta llegar a la capa de salida, donde entonces la salida será el resultado de las conexiones entre todas las entradas con todas las neuronas de todas las capas ocultas. Entonces, tomando en cuenta que una Feedforward Neural Network tiene como máximo dos capas, la salida debe estar dada por $a_1^{(3)} = \sigma(z_1^{(3)})$.

2. Funciones de Activación

Información recabada de (Galaxy Training Network, 2024)

- **Función de Activación Lineal:**
 - Se usa en la capa de salida para problemas de regresión.
 - No se utiliza en todas las capas, ya que en una red multicapa no permitiría modelar relaciones no lineales y se reduciría a una red de una sola capa.
- **Función de Activación Escalón Binario:**
 - Se usa en el perceptrón.
 - No es adecuada para redes multicapa, ya que su derivada es cero, impidiendo la actualización de pesos/biases durante la retropropagación.
- **Función Sigmoide:**
 - Se puede usar en capas ocultas y de salida.
 - Permite modelar relaciones no lineales.
 - Problema: el gradiente tiende a ser muy pequeño lejos del origen, lo que causa el problema del gradiente que desaparece, ralentizando el aprendizaje en redes profundas.
- **Función Tangente Hiperbólica (tanh):**
 - Similar a la Sigmoide, pero su rango es de -1 a 1.
 - Tiene derivadas más grandes, lo que reduce el problema del gradiente que desaparece en comparación con Sigmoide.
- **ReLU (Rectified Linear Unit):**
 - Popular en redes profundas, ya que no sufre el problema del gradiente que desaparece.
 - Es preferida sobre Sigmoide o tanh para redes con muchas capas.
 - Sigmoide o tanh aún se pueden usar en la capa de salida de redes profundas.
- **Sofmax:**
 - Usada en problemas de clasificación multiclase.
 - La suma de probabilidades de las capas anteriores dan como salidas una probabilidad igual a 1.
 - Ayuda a la interpretabilidad pero con limitaciones en clasificación binaria.

```
from IPython.display import Image, display
imagen = '/content/funciones.png'
display(Image(filename=imagen, width=600, height=300))
```



Name	Plot	Function, $f(x)$	Derivative of f , $f'(x)$	Range
Identity		x	1	$(-\infty, \infty)$
Binary step		$\begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$	$\begin{cases} 0 & \text{if } x \neq 0 \\ \text{undefined} & \text{if } x = 0 \end{cases}$	$\{0, 1\}$
Logistic, sigmoid, or soft step		$\sigma(x) = \frac{1}{1 + e^{-x}}$	$f(x)(1 - f(x))$	$(0, 1)$
tanh		$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$	$1 - f(x)^2$	$(-1, 1)$
Rectified linear unit (ReLU)		$\begin{cases} 0 & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}$	$\begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x > 0 \\ \text{undefined} & \text{if } x = 0 \end{cases}$	$[0, \infty)$

Figura 3. Distintos tipos de formas de funciones de activación y sus ecuaciones (Kamali, 2024)

```
from IPython.display import Image, display
imagen = '/content/softmax.png'
display(Image(filename=imagen, width=600, height=300))
```

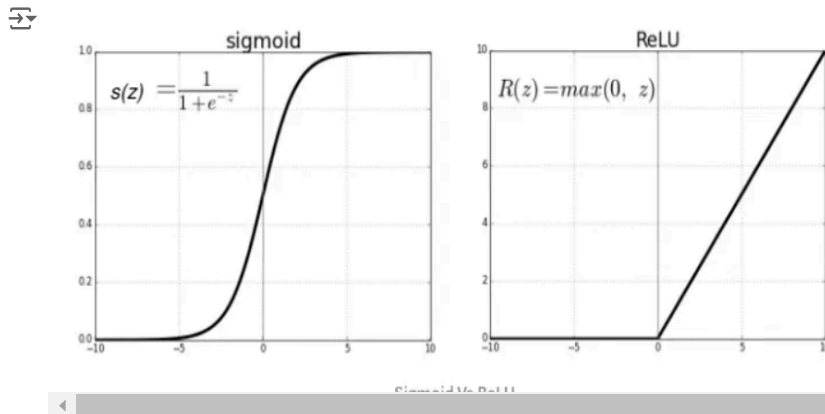


Figura 4. Ecuación y función de Softmax vs ReLU (Chaudhary, 2021)

(Se usó clasificación)

If the perceptron model will be used for classification, explain how it uses a decision boundary to classify inputs into two categories (binary classification).

- Dentro de una red neuronal (FNN) con múltiples entradas, cada entrada se combina con su respectivo peso, sumándose posteriormente con un término de sesgo. Este proceso se lleva a cabo en todas las capas de la red neuronal, lo que permite abordar problemas más complejos. Esto la hace funcional para problemas de clasificación, ya que los pesos y sesgos se ajustan durante el entrenamiento para definir los límites de decisión.
- La capacidad de manejar diferentes entradas permite a las FNN generar límites de decisión más precisos, logrando así distinguir entre varias clases.

3. Use case application

Discuss a real-world application where a perceptron model can be applied then use a public dataset that can be used as a use case for regression or classification.

El caso de aplicación que se desarrollará en este proyecto será una empresa recicladora de vidrio, la cual se dedica a vender este vidrio reciclado y procesado en placas o calcio (vidrio triturado) de vidrio. La manera en que lo clasifican para ajustarlo a su mercado potencial es a través del proceso de clasificación del departamento de Química. El proceso consiste en obtener la concentración química de 9 distintos elementos químicos y, con base en sus concentraciones, se asignan a una de las 5 categorías que se manejan.

Las 5 categorías se dividen dependiendo en el uso que se le puede dar al vidrio una vez trabajado, dependiendo la aplicación. Las categorías son:

1. Vidrio para ventanas para construcción
2. Vidrio para ventanas de vehículos
3. Vidrio para contenedores de vidrio
4. Vidrio para vajilla
5. Vidrio para luces delanteras de vehículos

- La página que utilizaremos para el desarrollo de este proyecto será Kaggle [8], una página con diversidad de bases de datos, esto porque se nos hizo más sencillo y la interfaz de la misma página se nos hizo más amigable.

Descripción de la base de datos

La base de datos que seleccionamos es sobre la identificación del tipo de cristal para diferentes aplicaciones dentro del uso de los mismos cristales, tal como lo son para edificios, vehículos, contenedores, lámparas y vajillas. [8]

La base de datos consta de 9 elementos químicos de los cuales, la concentración de dichos químicos en el cristal depende para qué puede ser utilizado el cristal. Las diferentes categorías son las siguientes:

- Id number: 1 to 214 (removed from CSV file)
- RI: refractive index
- Na: Sodium (unit measurement: weight percent in corresponding oxide, as are attributes 4-10)
- Mg: Magnesium
- Al: Aluminum
- Si: Silicon
- K: Potassium
- Ca: Calcium
- Ba: Barium
- Fe: Iron
- Type of glass: (class attribute)
 - 1 building_windows_float_processed
 - 2 building_windows_non_float_processed
 - 3 vehicle_windows_float_processed
 - 4 vehicle_windows_non_float_processed (none in this database)
 - 5 containers
 - 6 tableware
 - 7 headlamps

Por practicidad, se tomó la decisión de combinar las clases 1 y 2 de la base de datos, y eliminar por completo la base 4 (ya que en la base no existía ninguna muestra que cayera en esta categoría). De modo que al final nos resten 5 clases únicamente para mejorar el desempeño del modelo.

Aunado a esto y adelantándonos un poco a lo que se descubrió más adelante en el proyecto, se decidió aumentar la base de datos duplicando todos los datos ya existentes, agregando claro un ruido mínimo para evitar la duplicación exacta. Esto mejoró enormemente el desempeño del modelo, lo cual se discutirá más adelante en este documento.

4. Practical Implementation using Tensorflow in Python (Jupyter Notebook)

✓ 1. Dataset

Visualizar base de datos

```
#Librerias
import numpy as np# Importa la librería NumPy, útil para cálculos matemáticos y manejo de arreglos multidimensionales.
import pandas as pd # Importa la librería Pandas, que se usa para manipulación y análisis de datos.
import tensorflow as tf # Importa TensorFlow, una biblioteca para aprendizaje automático y redes neuronales.
import matplotlib.pyplot as plt # Importa Matplotlib para la creación de gráficos y visualización de datos.
from sklearn.model_selection import train_test_split # Importa la función train_test_split para dividir el conjunto de datos en entrenamient
```

```

from sklearn.preprocessing import StandardScaler # Escalador para normalizar los datos
import seaborn as sns # Importa Seaborn para la visualización de datos estadísticos.
from sklearn.metrics import classification_report, f1_score, confusion_matrix, ConfusionMatrixDisplay # Importa métricas de evaluación para

#Importamos los datos
from google.colab import drive # Importa la biblioteca google.colab.drive para acceder a la unidad de Google Drive en Colab.
drive.mount('/content/drive') # Monta la unidad de Google Drive en el entorno de Colab.

# Ruta al archivo CSV en tu Google Drive
file_path = '/content/drive/SharedDrives/Inteligencia/2do Parcial/glass.csv'
# Cargar el archivo CSV en un DataFrame
df = pd.read_csv(file_path) # Lee el archivo CSV y lo convierte en un DataFrame de Pandas.

# Combinar las clases 1 y 2 en una sola clase 1
df['Type'] = df['Type'].replace(2, 1)

# Eliminar todas las filas donde el tipo sea 4
df = df[df['Type'] != 4]

# Remapear las clases restantes para que sean numeradas secuencialmente del 1 al 5 con la función map
df['Type'] = df['Type'].map({1: 1, 3: 2, 5: 3, 6: 4, 7: 5})

# Aumentar el conjunto de datos duplicando las muestras con un pequeño ruido
def augment_data(df, noise_factor=0.01):
    # Separar las características y la variable objetivo
    x = df.drop(['Type'], axis=1).values
    y = df['Type'].values

    # Duplicar las muestras con ruido
    x_augmented = np.vstack([x, x + noise_factor * np.random.normal(loc=0.0, scale=1.0, size=x.shape)])
    y_augmented = np.concatenate([y, y])

    # Crear un nuevo DataFrame con los datos aumentados
    df_augmented = pd.DataFrame(x_augmented, columns=df.drop(['Type'], axis=1).columns)
    df_augmented['Type'] = y_augmented

    return df_augmented

# Aplicar el aumento de datos
df = augment_data(df)

# Verificar la forma del DataFrame aumentado
print("Shape:", df.shape) # Imprime las dimensiones del DataFrame, es decir, el número de filas y columnas.
df.head() # Muestra las primeras 5 filas del DataFrame para dar una vista previa de los datos.

```

Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
Shape: (428, 10)

	RI	Na	Mg	Al	Si	K	Ca	Ba	Fe	Type
0	1.52101	13.64	4.49	1.10	71.78	0.06	8.75	0.0	0.0	1
1	1.51761	13.89	3.60	1.36	72.73	0.48	7.83	0.0	0.0	1
2	1.51618	13.53	3.55	1.54	72.99	0.39	7.78	0.0	0.0	1
3	1.51766	13.21	3.69	1.29	72.61	0.57	8.22	0.0	0.0	1
4	1.51742	13.27	3.62	1.24	73.08	0.55	8.07	0.0	0.0	1

En esta tabla se puede apreciar el valor de las primeras 5 filas con sus respectivas columnas, lo que nos da una muestra de como viene la base de datos.

Además se aprecian los nombres de dichas columnas que son elementos químicos previamente comentados, así como el tipo de cristal.

Asimismo se aprecia la forma de (428,10) siendo el 428 el número de filas, y 10 el número de columnas.

2. Exploratory data analysis

Visualizar datos

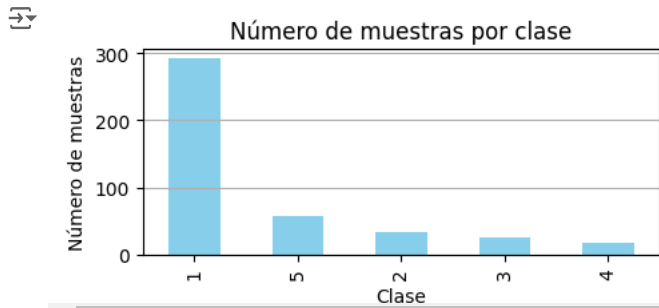
```

# Gráfico de barras para visualizar el número de muestras por clase
class_counts = df['Type'].value_counts()

# Crear el gráfico de barras
plt.figure(figsize=(5, 2))

```

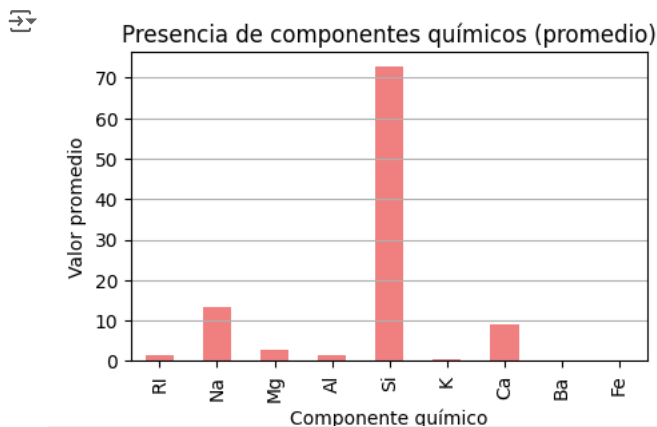
```
class_counts.plot(kind='bar', color='skyblue')
plt.title('Número de muestras por clase')
plt.xlabel('Clase')
plt.ylabel('Número de muestras')
plt.grid(axis='y')
plt.show()
```



Esta gráfica nos presenta de manera visual la distribución de los datos; vemos que existe un gran desbalance hacia una de las clases.

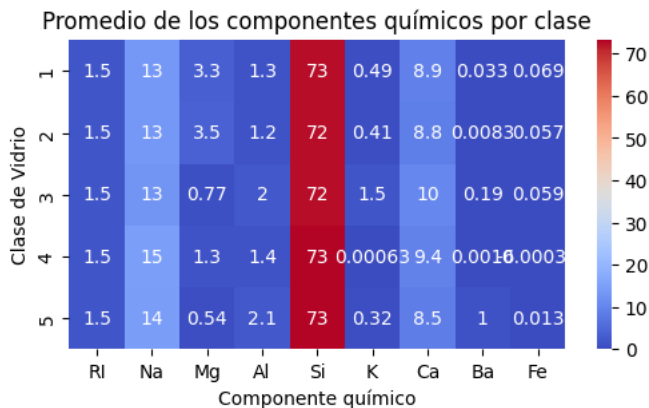
```
# Gráfico de presencia de componentes químicos (promedio de cada variable química)
component_means = df.drop('Type', axis=1).mean()
```

```
# Crear el gráfico de barras para los componentes químicos
plt.figure(figsize=(5, 3))
component_means.plot(kind='bar', color='lightcoral')
plt.title('Presencia de componentes químicos (promedio)')
plt.xlabel('Componente químico')
plt.ylabel('Valor promedio')
plt.grid(axis='y')
plt.show()
```



En esta gráfica de barras se puede apreciar que el componente químico que tiene una mayor presencia promedio en las muestras es el Silicio (Si), ya que se encuentra en más del 70% de las muestras, mientras que elementos como el Potasio (K), Bario (Ba) y Hierro (Fe) están prácticamente ausentes.

```
# Crear un heatmap de los promedios de los componentes químicos por clase
class_means = df.groupby('Type').mean()
plt.figure(figsize=(6, 3))
sns.heatmap(class_means, annot=True, cmap='coolwarm')
plt.title('Promedio de los componentes químicos por clase')
plt.xlabel('Componente químico')
plt.ylabel('Clase de Vidrio')
plt.show()
```

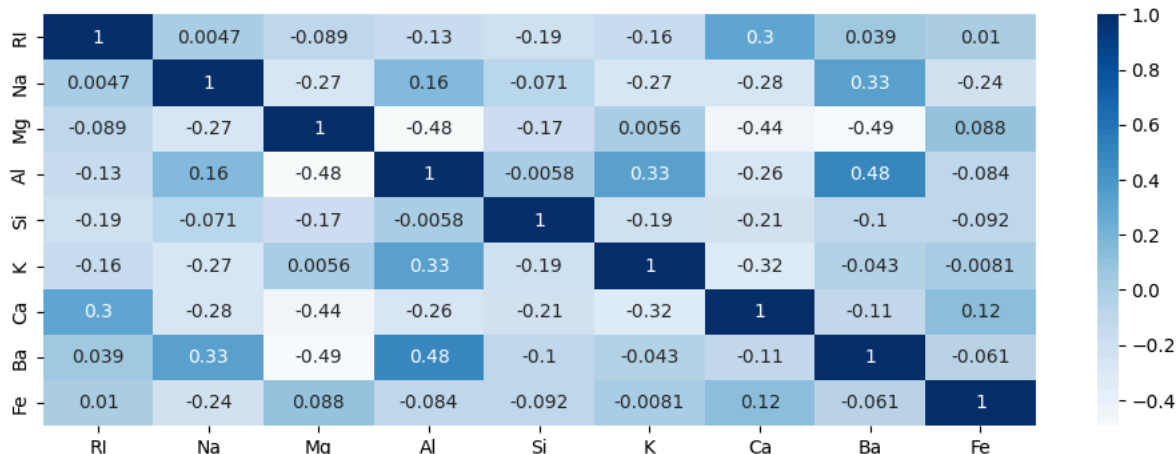

[+ Código](#)
[+ Texto](#)

Este heatmap muestra los promedios de los componentes químicos en cada clase de vidrio. El silicio (Si) es el componente más constante, con niveles similares (72-73) en todas las clases, lo que indica que es un ingrediente principal en todos los vidrios. El magnesio (Mg) es notablemente bajo en la clase 3, lo que podría ayudar a identificar esta clase, mientras que el bario (Ba) es mucho más alto en la clase 5, haciéndolo un buen indicador de esa clase. El calcio (Ca) también varía, siendo alto en las clases 2 y 4. Por otro lado, el sodio (Na) es bastante constante en todas las clases, y el hierro (Fe) está presente en cantidades muy pequeñas, por lo que estos dos componentes no parecen ser útiles para diferenciar las clases.

```
# Analizamos la correlación
import seaborn as sns # Importa Seaborn, una librería para visualización avanzada de datos.

# Eliminar la columna 'Type' antes de crear la matriz de correlación
corr_matrix = df.drop(['Type'], axis=1).corr()

# Crear el gráfico de la matriz de correlación
plt.figure(figsize=(12,4)) # Crea una figura con un tamaño específico de 15x5 pulgadas para mejorar la visibilidad del gráfico.
sns.heatmap(corr_matrix, annot=True, cmap=plt.cm.Blues) # Genera un mapa de calor de la matriz de correlación del DataFrame, con anotaciones
plt.show() # Muestra el gráfico en pantalla.
```



En esta matriz de correlación podemos observar cómo se relacionan las diferentes variables que describen las propiedades químicas del vidrio. Algunas relaciones notables incluyen una correlación moderada y positiva entre el bario (Ba) y el aluminio (Al), lo que indica que estos dos elementos tienden a aumentar juntos en la composición del vidrio. También hay una correlación negativa entre el magnesio (Mg) y el aluminio (Al), lo que sugiere que cuando el contenido de magnesio es alto, el aluminio tiende a ser bajo, y viceversa. El resto de las variables tienen correlaciones más débiles o cercanas a 0, lo que indica que no están fuertemente relacionadas entre sí. Esto sugiere que los diferentes componentes del vidrio no dependen fuertemente unos de otros en la mayoría de los casos, con algunas excepciones como las mencionadas. Sin embargo, las concentraciones de los distintos elementos juegan un papel fundamental a la hora de clasificar el vidrio por sus propiedades químicas y mecánicas.

3. Data preparation

```
# Continuar con los pasos anteriores
x = df.drop(['Type'], axis=1) # Eliminar la columna 'Type' del DataFrame y guardar el resto de las columnas en 'x'
```



```
y = df['Type'] - 1 # Restar 1 a los valores de 'Type' para que las etiquetas comiencen desde cero (indexación cero)

print("Shape de matriz x:", x.shape) # Muestra el tamaño de los datos de entrada, es decir, el número de filas y columnas de 'x'.
print("Shape de vector y:", y.shape) # Se imprime la forma de la variable objetivo 'y'.
```

```
↳ Shape de matriz x: (428, 9)
   Shape de vector y: (428,)
```

Shape de matriz x: (428, 9). Esto nos define que los datos de entrada (x) contienen 428 filas y 9 columnas.

Shape de vector y: (428,). Vector de etiquetas (y) tiene 428 filas.

✓ División de datos

```
# Escalar los datos (normalización)
scaler = StandardScaler()
x_scaled = scaler.fit_transform(x)

# Dividir los datos en conjuntos de entrenamiento y prueba
# El 80% de los datos se usa para entrenamiento (train) y el 20% para prueba (test).
x_train, x_test, y_train, y_test = train_test_split(x_scaled, y, test_size=0.2, random_state=42, stratify=y)
```

Se escalaron los datos para normalizar las características donde luego se dividieron los datos en entrenamiento y prueba.

Se utilizó el 80% de los datos para el entrenamiento (x_train, y_train) y el 20% para la prueba (x_test, y_test).

✓ 4. Model design and development

Clasificación

```
# Mostrar la forma de los conjuntos de datos para verificar la correcta separación
# Se imprimen la forma del conjunto de entrenamiento
print(f"x_train shape: {x_train.shape}, y_train shape: {y_train.shape}")
# Se imprimen la forma del conjunto de prueba
print(f"x_test shape: {x_test.shape}, y_test shape: {y_test.shape}")
```

```
↳ x_train shape: (342, 9), y_train shape: (342,)
   x_test shape: (86, 9), y_test shape: (86,)
```

Se muestran las dimensiones de los datos de entrenamiento y prueba.

x_train shape: (342, 9) y y_train shape: (342,). El conjunto de entrenamiento tiene 342 ejemplos con 9 características.

x_test shape: (86, 9) y y_test shape: (86,). El conjunto de prueba tiene 86 ejemplos con 9 características.

✓ Definir la arquitectura de la red neuronal

```
from tensorflow.keras.layers import BatchNormalization # Importa la capa de normalización por lotes (Batch Normalization).
def modelo2():
    model = tf.keras.Sequential([
        tf.keras.layers.InputLayer(input_shape=(x_train.shape[1],)), # Definir la capa de entrada con la forma de los datos de entrenamient
        tf.keras.layers.Dense(64, activation='relu'), # Capa densa con 64 neuronas y función de activación ReLU
        BatchNormalization(), # Capa de normalización por lotes para normalizar las salidas de la capa anterior
        tf.keras.layers.Dropout(0.25), # Capa de dropout con un 25% de tasa de abandono para evitar el sobreajuste
        tf.keras.layers.Dense(32, activation='relu'), # Otra capa densa con 32 neuronas y función de activación ReLU
        BatchNormalization(), # Otra capa de normalización por lotes para normalizar las salidas de la capa anterior
        tf.keras.layers.Dropout(0.25), # Otra capa de dropout con un 25% de tasa de abandono
        tf.keras.layers.Dense(5, activation='softmax') # Capa de salida con 5 neuronas (una para cada clase) y activación softmax
    ])
    return model # Retornar el modelo
```

✓ Imprime el resumen del modelo

```
# Creación del modelo.
model = modelo2() # Llamar a la función para obtener el modelo con dropout y normalización
model.summary() # Imprimir un resumen del modelo.
```

```

/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/input_layer.py:26: UserWarning: Argument `input_shape` is deprecated. Use
warnings.warn(

```

Model: "sequential_4"

Layer (type)	Output Shape	Param #
dense_12 (Dense)	(None, 64)	640
batch_normalization_8 (BatchNormalization)	(None, 64)	256
dropout_8 (Dropout)	(None, 64)	0
dense_13 (Dense)	(None, 32)	2,080
batch_normalization_9 (BatchNormalization)	(None, 32)	128
dropout_9 (Dropout)	(None, 32)	0
dense_14 (Dense)	(None, 5)	165

Total params: 3,269 (12.77 KB)
Trainable params: 3,077 (12.02 KB)
Non-trainable params: 192 (768.00 B)

Como salida se aprecia un resumen del modelo creado, donde se puede apreciar la diversas capas, uso de normalización y dropout.

dense_45: Capa densa con 64 neuronas.

batch_normalization_28: MMejorar la estabilidad del entrenamiento, reducir error y mejorar la velocidad de convergencia.

dropout_30: 25% de dropout para mejorar métricas de salida del modelo.

dense_46: Capa densa con 32 neuronas.

batch_normalization_29: Mejorar la estabilidad del entrenamiento, reducir error y mejorar la velocidad de convergencia.

dropout_31: 25% de dropout para mejorar métricas de salida del modelo.

dense_47: Capa densa con 5 neuronas.

Total params: 3,269 (12.77 KB): Parámetros totales.

Trainable params: 3,077 (12.02 KB): Parámetros ajustables durante el entrenamiento.

Non-trainable params: 192 (768.00 B): Parámetros de la normalización no ajustables durante el entrenamiento.

```

# Compilar el modelo usando 'sparse_categorical_crossentropy'
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

```

```

# Entrenar el modelo
history = model.fit(x_train, y_train, epochs=150, batch_size=26, validation_split=0.15)

```

```

Epoch 1/150
12/12 ————— 3s 34ms/step - accuracy: 0.2682 - loss: 1.8646 - val_accuracy: 0.2115 - val_loss: 1.5557
Epoch 2/150
12/12 ————— 0s 7ms/step - accuracy: 0.3031 - loss: 1.8457 - val_accuracy: 0.5000 - val_loss: 1.4475
Epoch 3/150
12/12 ————— 0s 7ms/step - accuracy: 0.4756 - loss: 1.4679 - val_accuracy: 0.6731 - val_loss: 1.3427
Epoch 4/150
12/12 ————— 0s 8ms/step - accuracy: 0.5142 - loss: 1.2685 - val_accuracy: 0.7308 - val_loss: 1.2409
Epoch 5/150
12/12 ————— 0s 9ms/step - accuracy: 0.5502 - loss: 1.2471 - val_accuracy: 0.7308 - val_loss: 1.1466
Epoch 6/150
12/12 ————— 0s 9ms/step - accuracy: 0.6263 - loss: 1.0560 - val_accuracy: 0.7500 - val_loss: 1.0644
Epoch 7/150
12/12 ————— 0s 8ms/step - accuracy: 0.6306 - loss: 1.0590 - val_accuracy: 0.7500 - val_loss: 0.9923
Epoch 8/150
12/12 ————— 0s 10ms/step - accuracy: 0.6745 - loss: 0.9317 - val_accuracy: 0.7500 - val_loss: 0.9219
Epoch 9/150
12/12 ————— 0s 7ms/step - accuracy: 0.7856 - loss: 0.8051 - val_accuracy: 0.7692 - val_loss: 0.8575
Epoch 10/150
12/12 ————— 0s 11ms/step - accuracy: 0.7740 - loss: 0.7681 - val_accuracy: 0.7692 - val_loss: 0.8098
Epoch 11/150
12/12 ————— 0s 10ms/step - accuracy: 0.7728 - loss: 0.7615 - val_accuracy: 0.7885 - val_loss: 0.7642
Epoch 12/150
12/12 ————— 0s 9ms/step - accuracy: 0.8179 - loss: 0.5979 - val_accuracy: 0.8077 - val_loss: 0.7248
Epoch 13/150
12/12 ————— 0s 6ms/step - accuracy: 0.7779 - loss: 0.6935 - val_accuracy: 0.8077 - val_loss: 0.6969
Epoch 14/150
12/12 ————— 0s 6ms/step - accuracy: 0.7816 - loss: 0.6687 - val_accuracy: 0.8077 - val_loss: 0.6727
Epoch 15/150

```

```

12/12 ————— 0s 5ms/step - accuracy: 0.8149 - loss: 0.5996 - val_accuracy: 0.8077 - val_loss: 0.6508
Epoch 16/150
12/12 ————— 0s 5ms/step - accuracy: 0.8282 - loss: 0.5647 - val_accuracy: 0.8077 - val_loss: 0.6315
Epoch 17/150
12/12 ————— 0s 5ms/step - accuracy: 0.8624 - loss: 0.5146 - val_accuracy: 0.8077 - val_loss: 0.6171
Epoch 18/150
12/12 ————— 0s 6ms/step - accuracy: 0.7532 - loss: 0.7607 - val_accuracy: 0.8077 - val_loss: 0.6011
Epoch 19/150
12/12 ————— 0s 5ms/step - accuracy: 0.8075 - loss: 0.6165 - val_accuracy: 0.8077 - val_loss: 0.5915
Epoch 20/150
12/12 ————— 0s 5ms/step - accuracy: 0.8285 - loss: 0.6744 - val_accuracy: 0.8077 - val_loss: 0.5844
Epoch 21/150
12/12 ————— 0s 5ms/step - accuracy: 0.8638 - loss: 0.5562 - val_accuracy: 0.8077 - val_loss: 0.5757
Epoch 22/150
12/12 ————— 0s 6ms/step - accuracy: 0.8535 - loss: 0.5036 - val_accuracy: 0.8077 - val_loss: 0.5658
Epoch 23/150
12/12 ————— 0s 5ms/step - accuracy: 0.8517 - loss: 0.4957 - val_accuracy: 0.8077 - val_loss: 0.5542
Epoch 24/150
12/12 ————— 0s 7ms/step - accuracy: 0.8445 - loss: 0.5511 - val_accuracy: 0.8077 - val_loss: 0.5411
Epoch 25/150
12/12 ————— 0s 4ms/step - accuracy: 0.8077 - loss: 0.5742 - val_accuracy: 0.8269 - val_loss: 0.5302
Epoch 26/150
12/12 ————— 0s 5ms/step - accuracy: 0.8219 - loss: 0.5400 - val_accuracy: 0.8269 - val_loss: 0.5243
Epoch 27/150
12/12 ————— 0s 6ms/step - accuracy: 0.8464 - loss: 0.4900 - val_accuracy: 0.8269 - val_loss: 0.5179
Epoch 28/150
12/12 ————— 0s 5ms/step - accuracy: 0.8656 - loss: 0.4698 - val_accuracy: 0.8077 - val_loss: 0.5106
Epoch 29/150
12/12 ————— 0s 6ms/step - accuracy: 0.8371 - loss: 0.5069 - val_accuracy: 0.8077 - val_loss: 0.5064

```

Compilación y entrenamiento del modelo.

✓ Evaluar el modelo en el conjunto de prueba

```

# Evaluamos el modelo con los datos de prueba, y nos entrega la perdida y el accuracy
loss, accuracy = model.evaluate(x_test, y_test)
#Imprimimos el accuracy con 4 valores decimales.
print(f"Test Accuracy: {accuracy:.4f}")

```

```

3/3 ————— 0s 5ms/step - accuracy: 0.8950 - loss: 0.3299
Test Accuracy: 0.8837

```

El modelo tiene una precisión de 0.8837 y una perdida de 0.3299 con una presión en la prueba de 0.8837. Esto es un indicador claro de que el modelo ha aprendido de manera correcta y el modelo tiene buena precisión.

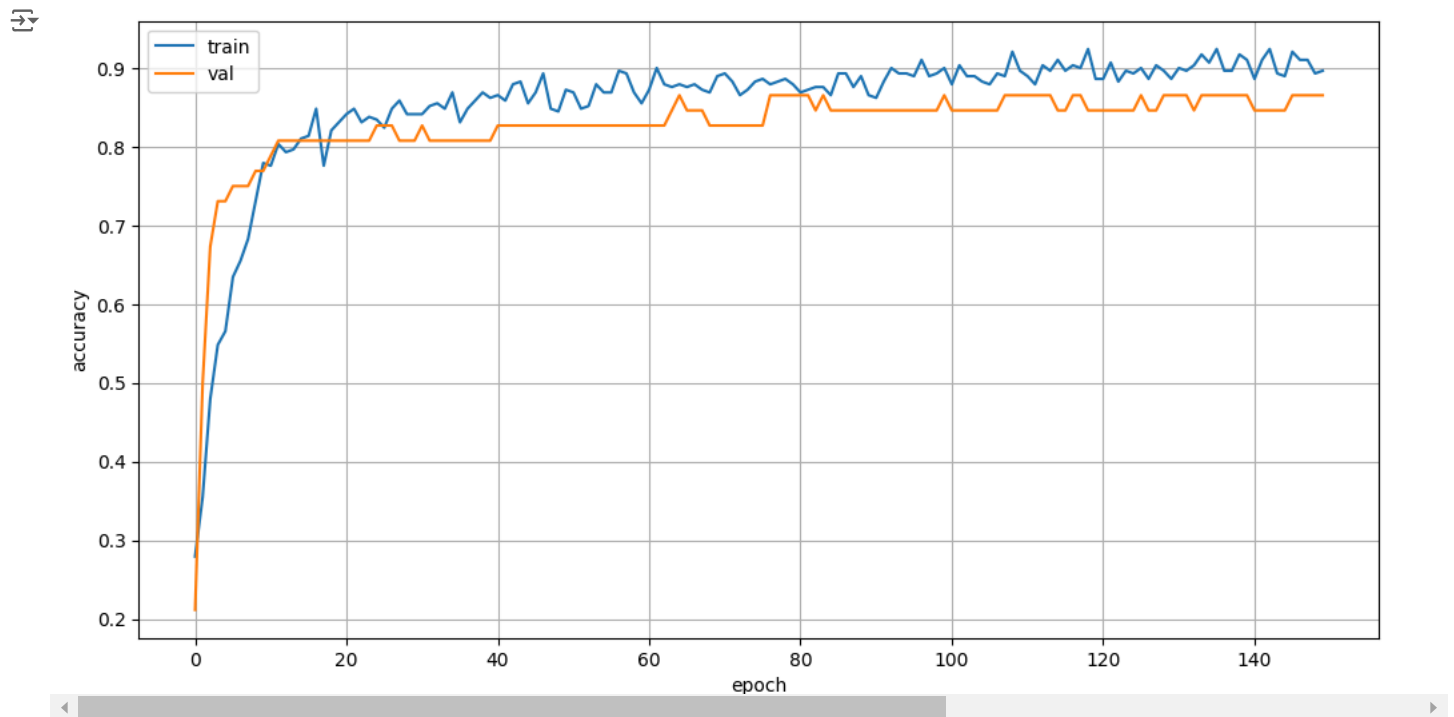
✓ Graficar

```

# Graficamos el comportamiento de los datos de train y test

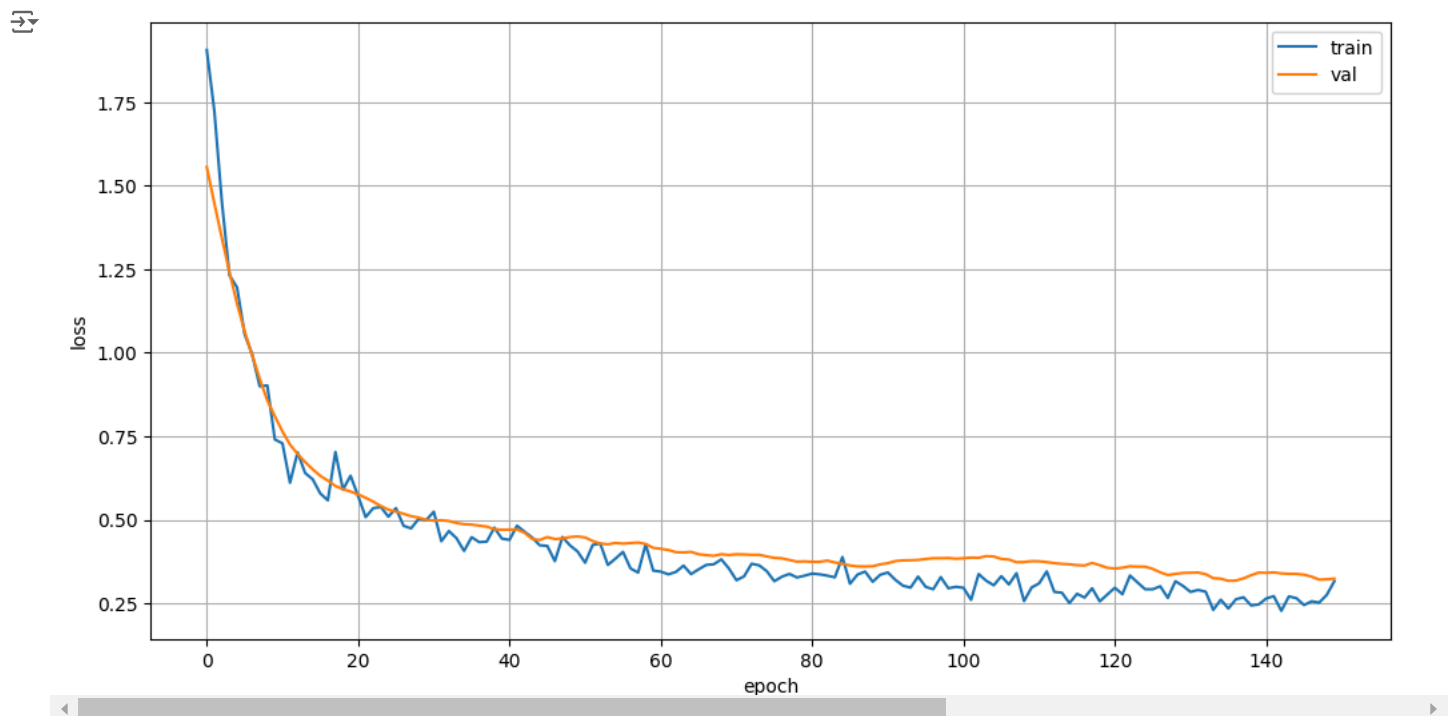
plt.figure(figsize=(12,6)) #Tamaño.
plt.plot(history.history['accuracy']) # Gráfica de la exactitud del conjunto de entrenamiento.
plt.plot(history.history['val_accuracy']) # Gráfica de la exactitud del conjunto de validación.
plt.xlabel('epoch') # Etiqueta del eje x.
plt.ylabel('accuracy') # Etiqueta del eje y.
plt.legend(['train', 'val']) # Leyenda para distinguir entrenamiento y validación.
plt.grid(); # Agregar una cuadrícula para mayor claridad en la gráfica.

```



La gráfica muestra que la precisión tanto en el conjunto de entrenamiento como en el de validación mejora rápidamente en las primeras 20 épocas. Después de eso, ambas se estabilizan. El entrenamiento alcanza cerca del 90% de precisión, mientras que la validación se queda alrededor del 80-85%. Esto indica que el modelo está aprendiendo bien, pero la diferencia entre ambas curvas sugiere que podría haber un ligero sobreajuste, aunque el modelo aún generaliza bastante bien.

```
# Graficamos el comportamiento de la función de pérdida de los datos de train y test
plt.figure(figsize=(12,6))
plt.plot(history.history['loss']) # Gráfica de la pérdida del conjunto de entrenamiento.
plt.plot(history.history['val_loss']) # Gráfica de la pérdida del conjunto de validación.
plt.xlabel('epoch') # Etiqueta del eje x.
plt.ylabel('loss') # Etiqueta del eje y.
plt.legend(['train', 'val']) # Leyenda para distinguir entrenamiento y validación.
plt.grid(); # Agregar una cuadrícula para mayor claridad en la gráfica.
```



Esta gráfica de la pérdida muestra que tanto la pérdida en el conjunto de entrenamiento como en el de validación disminuyen rápidamente al principio, especialmente en las primeras 20 épocas. Después de eso, ambas curvas se estabilizan. La pérdida en el entrenamiento sigue disminuyendo gradualmente, mientras que la de validación se estabiliza alrededor de 0.4. Esto sugiere que el modelo está aprendiendo bien. Existe una separación entre ambas líneas, sin embargo, ambas pérdidas se mantienen bastante cercanas, lo que es una señal positiva.

▼ Imprimir los pesos del modelo

```
#Imprimimos los pesos.
weights = model.layers[0].get_weights() # Obtener los pesos de la primera capa (capa de entrada) del modelo entrenado.
print(weights) # Imprimir los pesos de la capa de entrada.
```

```
[array([[ 7.73881748e-02,  1.74638599e-01,  5.24052419e-02,
         8.33515003e-02,  2.13288113e-01, -2.20153540e-01,
         2.71825075e-01,  4.68030907e-02,  1.18293330e-01,
         1.74802113e-02,  3.62347029e-02,  5.03308140e-02,
        -4.29930873e-02, -1.60326585e-01,  1.47954822e-01,
        -2.12622881e-01,  1.57907829e-01,  1.59827396e-01,
        -2.92777836e-01, -2.14325547e-01,  1.24421902e-01,
         3.06984991e-01, -4.87596635e-03, -9.43650454e-02,
         6.01701178e-02, -2.80706704e-01, -6.18837308e-04,
         1.93879753e-02, -6.21806644e-03, -3.25981639e-02,
        -3.74892615e-02,  5.56334555e-02,  1.67818427e-01,
        -4.51187082e-02,  1.02132313e-01, -1.95690736e-01,
         1.83741823e-01,  5.56677841e-02,  3.18813957e-02,
         1.98614392e-02,  5.45149632e-02, -2.32273310e-01,
         2.53997110e-02,  2.33922869e-01, -4.41577993e-02,
        -1.15378581e-01,  2.08522096e-01,  2.74336059e-02,
        -9.92765576e-02,  2.44436413e-01,  2.09103689e-01,
         6.25229767e-03,  1.63454771e-01, -1.08619258e-01,
         1.70700382e-02, -3.97860743e-02,  2.29985222e-01,
         2.66605578e-02,  2.21082550e-02, -1.38643354e-01,
         1.47074848e-01,  1.48372009e-01,  1.07724652e-01,
        -1.58665612e-01],
 [-1.52474403e-01,  9.50062796e-02, -3.13139796e-01,
        -3.18409026e-01,  1.88681677e-01, -1.61554381e-01,
        -7.42196664e-02,  1.38100699e-01,  2.19505996e-01,
        -1.40189230e-01, -2.62970209e-01,  1.60006493e-01,
         1.72859371e-01,  1.04649708e-01, -4.15014056e-03,
        -2.99547017e-01, -1.12776481e-01,  2.20825881e-01,
        -2.44283661e-01,  2.57217854e-01,  1.44723311e-01,
         2.58656070e-02,  1.05751595e-02,  1.65282860e-01,
        -2.18429163e-01, -4.69087576e-03, -1.55802205e-01,
        -8.62238482e-02, -8.37644935e-02, -2.08285943e-01,
        -1.00832291e-01, -5.48440702e-02,  3.17442045e-02,
         1.15913404e-02, -1.89674214e-01, -2.13860944e-01,
         8.69226605e-02,  5.15535250e-02, -2.58814991e-01,
         1.39354452e-01, -5.87193295e-02, -1.23153657e-01,
         2.61431783e-01,  1.28882319e-01, -1.19271338e-01,
         1.36592656e-01,  2.22140267e-01,  7.96168074e-02,
         4.87301685e-02, -1.93320855e-01,  5.05848005e-02,
         3.66276354e-02, -4.25307192e-02, -1.05367281e-01,
        -1.99004963e-01,  2.40842383e-02, -2.74321344e-02,
        -8.18928406e-02, -2.24325567e-01,  2.62716055e-01,
        -9.12148878e-02,  5.55704758e-02,  1.00164954e-02,
         3.17129612e-01],
 [-5.02756424e-02,  2.35116437e-01, -3.70485447e-02,
         1.63289368e-01, -2.60567158e-01,  3.05923194e-01,
         2.44229529e-02,  2.55796015e-01,  2.75201708e-01,
         2.99050093e-01,  1.60640389e-01, -1.75140902e-01,
         3.25188786e-01,  1.58924237e-01,  1.54619560e-01,
        -1.36208862e-01,  3.00628692e-01,  2.10315391e-01,
        -1.28956348e-01, -1.69512451e-01,  1.40708670e-01,
         6.93100989e-02,  1.34755418e-01, -6.19417727e-02,
         1.68737099e-01, -1.01093508e-01, -1.30008668e-01,
         3.63335073e-01, -3.80498208e-02, -1.91514082e-02,
        -1.17793240e-01, -1.51977181e-01,  1.46383539e-01,
         1.04418240e-01, -3.08059864e-02,  2.41086558e-01,
         1.42793208e-01,  5.11669368e-02, -1.46210864e-01,
         2.05432698e-01, -6.21814607e-03,  2.47677580e-01,
```

La salida son los pesos de la primera capa densa del modelo.

Estos pesos representan la conexión entre las neuronas de entrada y las neuronas ocultas.

Cada peso almacenado en el arreglo nos determina la importancia de cada entrada en el proceso de predicción.

✓ Evaluar el modelo y mostrar la exactitud

```
from sklearn.metrics import classification_report, f1_score

# Evaluar el modelo
loss, accuracy = model.evaluate(x_test, y_test)# Evaluar el modelo
print(f"Test Accuracy: {accuracy:.4f}")

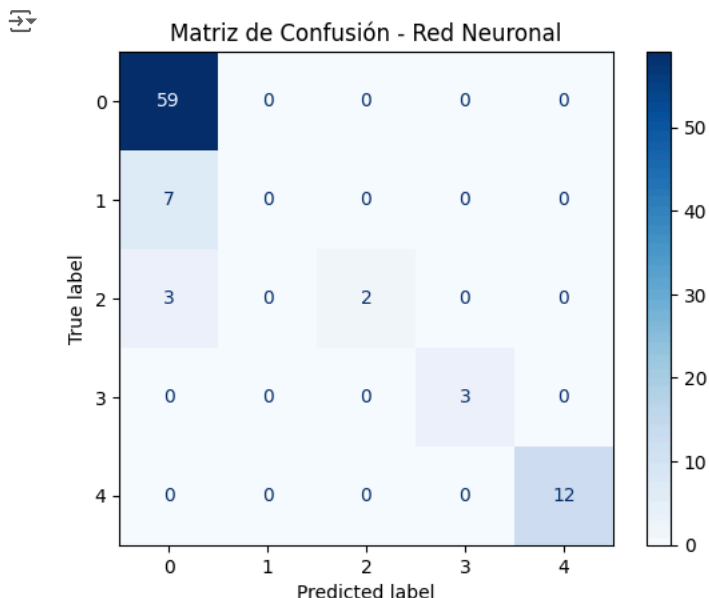
# Predecir las clases para el conjunto de prueba
y_pred_nn = model.predict(x_test) # Generar predicciones para los datos de prueba.
y_pred_nn_classes = np.argmax(y_pred_nn, axis=1) # Convertir las predicciones en las clases predichas.
```

3/3 ————— 0s 4ms/step - accuracy: 0.8950 - loss: 0.3299
 Test Accuracy: 0.8837
 WARNING:tensorflow:5 out of the last 7 calls to <function TensorFlowTrainer.make_predict_function.<locals>.one_step_on_data_distributed
 1/3 ————— 0s 93ms/stepWARNING:tensorflow:6 out of the last 9 calls to <function TensorFlowTrainer.make_predict_function.<locals>.one_step_on_data_distributed
 3/3 ————— 0s 39ms/step

El modelo tiene una precisión aceptable en el conjunto de prueba, con una pérdida razonablemente baja. Esto nos da seguridad respecto a la aproximación de las predicciones con las clases esperadas en su mayoría.

✓ Matriz de confusión

```
# Mostrar la matriz de confusión
c_matrix = confusion_matrix(y_test, y_pred_nn_classes) # Se genera la matriz de confusión
disp = ConfusionMatrixDisplay(confusion_matrix=c_matrix, display_labels=np.unique(y_test)) # Visualización de la matriz de confusión
disp.plot(cmap=plt.cm.Blues)
plt.title('Matriz de Confusión - Red Neuronal') # Se establece el título del gráfico
plt.show() # Se muestra el gráfico en pantalla.
```



Esta gráfica de la pérdida muestra que tanto la pérdida en el conjunto de entrenamiento como en el de validación disminuyen rápidamente al principio, especialmente en las primeras 20 épocas. Después de eso, ambas curvas se estabilizan. La pérdida en el entrenamiento sigue disminuyendo gradualmente, mientras que la de validación se estabiliza alrededor de 0.4. Esto sugiere que el modelo está aprendiendo bien. Existe una separación entre ambas líneas, sin embargo, ambas pérdidas se mantienen bastante cercanas, lo que es una señal positiva.

✓ Cálculo de métricas de desempeño

```
from sklearn.metrics import classification_report
print(classification_report(y_test,y_pred_nn_classes))
```

	precision	recall	f1-score	support
0	0.86	1.00	0.92	59
1	0.00	0.00	0.00	7
2	1.00	0.40	0.57	5
3	1.00	1.00	1.00	3
4	1.00	1.00	1.00	12
accuracy			0.88	86
macro avg	0.77	0.68	0.70	86
weighted avg	0.82	0.88	0.84	86

```

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and be
_warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and be
_warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and be
_warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))

```

Explicación de los resultados

Precisión: Es la división de los verdaderos positivos entre la suma de los verdaderos positivos y los falsos positivos. Es decir, calcula cuántos de los valores categorizados como positivos lo fueron de manera correcta. Entonces, para la clase cero, se encontraron 68 valores, de los cuales 58 fueron categorizados de manera correcta, pero 10 no lo fueron (7 se clasificaron como clase 1 y 3 como clase 2). Después de esto, encontramos datos algo atípicos para las demás clases; sin embargo, tienen fundamento en la naturaleza de la base, siendo que para la clase 1, con una precisión de 0, tenemos solo 7 datos, y ninguno fue clasificado de manera adecuada, generando una división de 0/1, por lo que se representa como 0. Caso contrario ocurre con las clases 2, 3 y 4, donde el modelo fue capaz de encontrar todos los datos de manera adecuada.

Recall (sensibilidad): Es la división de los verdaderos positivos entre todos los valores que realmente pertenecen a esa categoría. Por ejemplo, para la clase 0, tenemos 58 verdaderos positivos y un falso negativo que se clasificó como 0 cuando realmente debió haber sido 1, dándonos un valor de recall para la clase 0 de 0.98 (58/59). Nuevamente, tenemos un valor de 0 en la categoría 1, ya que tenemos un numerador de 0. Para la categoría 2, tenemos 3 falsos negativos que se categorizaron como 0, por lo que tenemos un recall de 0.40 (2/5). Para las categorías 3 y 4, los valores de recall son 1, ya que el modelo categorizó de manera correcta todos los valores correspondientes.

F1 Score: Unifica los valores de recall y precisión, y se calcula multiplicando por 2 la división entre la multiplicación de precisión por recall y la suma de ambos.

Support: Únicamente nos indica el número de valores en el dataset por cada una de las muestras.

Después tenemos la precisión global, que nos indica en qué porcentaje el modelo está categorizando correctamente los datos. En este caso, por ejemplo, tenemos 75 verdaderos positivos de un total de 86 valores, por lo que 75/86 nos da un total de 0.87, lo que indica que el modelo tiene un 87% de probabilidad de categorizar los datos de manera correcta.

Macro AVG: Calcula el promedio de los F1 scores. En este caso, 3.59/5 nos da un total aproximado de 0.70. Sin embargo, debido a que nuestra base de datos está desbalanceada, utilizamos mejor el weighted avg, que otorga un peso a las clases dependiendo del número de datos que contienen, dándole así mayor relevancia a sus F1 scores.

Discussions and Conclusions

1. Discuss about the limitations of a deep neural network in complex tasks

Las DNN requieren grandes cantidades de datos para entrenarse de manera efectiva, lo que puede ser un problema en conjuntos de datos pequeños, como en nuestro proyecto, donde tuvimos que duplicar los datos con ruido para mejorar el desempeño. También son propensas al sobreajuste, aprendiendo demasiado de los detalles específicos de los datos de entrenamiento, lo que las hace menos efectivas con datos nuevos; en nuestro caso, mejoramos esto usando Dropout. Además, entrenarlas puede ser muy costoso en términos de tiempo y recursos computacionales, especialmente en tareas complejas. Finalmente, son difíciles de interpretar, ya que funcionan como "cajas negras" mientras más se van haciendo más complejas, lo que complica la comprensión de cómo es que toman decisiones.

2. Was the perceptron model able to capture the relationship between inputs and outputs in your use case application?

Sí, el modelo fue capaz de capturar la relación entre las entradas (los componentes químicos del vidrio) y las salidas (las clases de vidrio). Esto se observó en la precisión relativamente alta que alcanzó tanto en el conjunto de entrenamiento como en el de validación. Al aplicar técnicas como Batch Normalization y Dropout, logramos mejorar la generalización del modelo y evitar el sobreajuste. Además, al duplicar el conjunto de

datos con un pequeño ruido, mejoramos significativamente su capacidad para aprender las relaciones entre los datos, especialmente dado que nuestra base de datos original era pequeña. Aunque el modelo mostró un buen rendimiento, aún presentó algunas dificultades para clasificar correctamente ciertas clases, lo que sugiere que podría beneficiarse de más datos o ajustes adicionales.

3. Was the perceptron model able to learn faster for larger values of the learning rate?

A Si, sin embargo bajaba el accuracy, por lo que probamos con diversos valores, desde el 0.0001 hasta el 0.1 y encontramos lo siguiente. LR = 0.0001 - Accuracy = 0.83 LR = 0.001 - Accuracy = 0.84 LR = 0.1 - Accuracy = 0.83 LR = Default (0.01) - Accuracy = 0.88

4. Was it necessary to normalise/scale the predictor variables? why?

Sí, fue necesario normalizar o escalar las variables predictoras en este proyecto. Esto se debe a que las redes neuronales son sensibles a las escalas de los datos. En nuestro caso, los componentes químicos del vidrio tienen rangos muy diferentes. Si no escaláramos los datos, las variables con valores más grandes podrían dominar el proceso de aprendizaje, haciendo que el modelo se concentre más en ellas y menos en las demás. Al normalizar las variables, garantizamos que todas tengan el mismo peso en el entrenamiento, lo que mejora la eficiencia y el rendimiento del modelo.

5. What strategies did you follow to tune the training hyperparameters?

Para afinar los hiperparámetros del entrenamiento, seguimos un enfoque donde probamos diferentes combinaciones de valores para encontrar la configuración que mejor se ajustara al modelo. En particular, experimentamos con distintas tasas de aprendizaje, tamaños de batch size, números de épocas, y tasas de dropout. Evaluamos el rendimiento del modelo con cada ajuste utilizando el conjunto de validación y analizamos las métricas de precisión y pérdida para identificar los valores que ofrecían un equilibrio óptimo entre el rendimiento del modelo y la generalización. A lo largo de este proceso, encontramos que el uso de un tamaño de lote de 26, 150 épocas, y una tasa de Dropout del 25% proporcionaron los mejores resultados. Esta experimentación nos permitió optimizar el modelo de manera sistemática.

6. Did the learning process always converge? if not, what was the reason why it failed to do so?

No, fue necesario hacer algunas modificaciones a la base de datos, como el duplicar la cantidad de muestras para que el modelo pudiera converger. Esto se debió a que la base no era lo suficientemente robusta como para generar un modelo de clasificación de 7 clases, ergo también redujimos las clases a 5. Además el usar tantos métodos y funciones más sofisticadas como Dropout, la cual con una base muy pequeña, lejos de ayudarla, provoca underfitting.

7. What improvement can be adopted to solve the use case application more efficiently and robustly? e.g., using a multi-layer network, why?

A En este caso, hicimos algunas otras pruebas sobre la arquitectura de la Red Neuronal, por ejemplo, el agregar una capa extra, aumentar/disminuir el número de neuronas por capa y realizar iteraciones con diversos valores de dropout, sin embargo, el cambio en los parámetros del modelo fue casi nulo, por lo que concluimos que el problema en este modelo proviene directamente de la base, cómo se mencionó anteriormente, la base de datos tienen un muy notorio desbalance en las clases, pero a su vez, este desbalance se entiende normal bajo el contexto de la base, es decir, si pensamos en una empresa recicladora de vidrios, podemos suponer que recibirán en mucho mayor volumen vidrios que provengan de artículos cotidianos cómo lo pueden ser envases de refresco, en comparación a vidrios utilizados en objetos menos convencionales, cómo lo pudieran ser por ejemplo los vidrios de los utilizados en balcones, lo que justifica el desbalance de las clases; además de este desbalance, la base contiene un número de registros, que si bien, suficientes, retador para trabajarla.

Tomando en cuenta lo anterior las mejoras que proponemos son las siguientes: Solicitar al cliente una medición más constante que alimente de manera adecuada la base de datos. Aplicar SMOTE o SMOTEEN a la base de datos (Se descarta el uso de RUS por el número limitado de muestras).

8. Provide personal conclusions about what you have achieved in this project.

Víctor:

En este proyecto, utilizamos una red neuronal para clasificar tipos de vidrio según su composición química. Entrenamos un modelo capaz de identificar a qué tipo de vidrio pertenece una muestra en función de la concentración de diversos componentes. Este tipo de modelo puede ser útil en la industria del vidrio, donde es importante clasificarlo con precisión para su aplicación óptima. Apreciamos la importancia de normalizar los datos, ya que las distintas escalas de los componentes químicos afectan el cómo aprende el modelo. También implementamos Batch Normalization y Dropout para evitar el sobreajuste y mejorar la generalización. Además, duplicar el conjunto de datos con un pequeño ruido ayudó mucho, dado que el conjunto original era muy pequeño. Finalmente, usamos gráficos de precisión y pérdida para visualizar el rendimiento y ajustar los hiperparámetros, lo que nos permitió mejorar el desempeño del modelo de manera efectiva a la hora de estar ajustando detalles. En general, cada vez aprendo más acerca de cómo realmente estructurar una red y el cómo doblegarla a nuestras necesidades, así como el adaptarla a las condiciones, como lo es el tipo de base de datos que se tiene.

Virgilio:

El desarrollo de este proyecto me ayudó a poner en práctica mis conocimientos adquiridos a lo largo del curso donde se destaca el uso de batch normalización, drop out, y el movimiento de los hiperparámetros lo cual fue clave para el buen desempeño del modelo. Durante el desarrollo del modelo tuvimos problemas referentes a que el desempeño del modelo estaba muy bajo con pérdidas considerablemente grandes, por lo que al mover los hiperparámetros aumentando a 150 las épocas y bajando el batch size a 26 el modelo se desempeñó de mejor manera. Para la obtención de dichos parámetros se tuvo que realizar un ajuste fino, para obtener la mejor respuesta. Consecuentemente se duplicaron los datos, lo que generó el resultado final. Sin duda alguna el uso de batch normalization, y drop out, fueron piezas clave para obtener un buen resultado. Por lo tanto, el uso de estas funciones, en efecto, si ayuda al entrenamiento del modelo aunque no se debe de exceder de la función de drop out porque si no se puede llegar a un underfitting, o afectar el rendimiento del modelo. Finalmente el proyecto si fue de importancia en la comprensión de los nuevos conceptos, y espero poder seguir aprendiendo más funciones referentes a inteligencia artificial.

Marcelo:

Considero que este proyecto fue crucial para entender realmente el impacto que tiene las modificaciones que se le pueden realizar a la arquitectura de los modelos, así como herramientas como el dropout o la normalización, personalmente, disfrute mucho la base de datos que utilizamos ya que creo que puede escalarse a la industria recicladora y ser un excelente proyecto, además de que si bien he de admitir que no fue para nada la base de datos más sencilla de trabajar especialmente por su desbalance y su tamaño limitado de observaciones, considero que son el tipo de problemas al que nos vamos a enfrentar una vez comencemos a buscar clientes para realizar proyectos, por lo que considero es importante retornar a nosotros mismos a trabajar con bases de datos que si bien difíciles, realistas. Considero que el resultado que obtuvimos es bueno, pero cómo todo, puede mejorar, quizás ahorita no conozco aún las herramientas para hacerlo, pero termino este

✓ Referencias

Banoula, M. (2024, August 19). What is Perceptron: A Beginners Guide for Perceptron. Simplilearn.com.

<https://www.simplilearn.com/tutorials/deep-learning-tutorial/perceptron>

Benardos, P., & Vosniakos, G. (2006). Optimizing feedforward artificial neural network architecture. Engineering Applications of Artificial Intelligence, 20(3), 365-382. <https://doi.org/10.1016/j.engappai.2006.06.005>

Brownlee, J. (2020, August 5). Perceptron algorithm for classification in Python. MachineLearningMastery.com.

<https://machinelearningmastery.com/perceptron-algorithm-for-classification-in-python/>

Chaudhary, M. (2021, 15 diciembre). Activation functions: Sigmoid, TANH, ReLU, Leaky ReLU, SoftMax. Medium.

<https://medium.com/@cmukesh8688/activation-functions-sigmoid-tanh-relu-leaky-relu-softmax-50d3778dcea5>

Ciberseg. (2021, November 16). Perceptron, qué es y cómo se usa en Machine Learning. Ciberseguridad.

<https://ciberseguridad.com/guias/nuevas-tecnologias/machine-learning/perceptron/>

DeepAI. (2020, 25 junio). Hidden layer. DeepAI. <https://deepai.org/machine-learning-glossary-and-terms/hidden-layer-machine-learning>

Galaxy Training Network. (2024, June 14). Statistics and machine learning / Deep Learning (Part 1) - Feedforward neural networks (FNN) /

Hands-on: Deep Learning (Part 1) - Feedforward neural networks (FNN). <https://training.galaxyproject.org/training-material/topics/statistics/tutorials/FNN/tutorial.html>

Glass Classification. (2017, January 27). <https://www.kaggle.com/datasets/uciml/glass>

Haykin, S. S. (1999). Neural networks: A Comprehensive Foundation. Upper Saddle River, N.J.: Prentice Hall.

ICHI.PRO. (n.d.). ¿Qué es un perceptrón? - Conceptos básicos de las redes neuronales. <https://ichi.pro/es/que-es-un-perceptron-conceptos-basicos-de-las-redes-neuronales-216397265290640>

Kamali, K. (2024, 14 junio). Statistics and machine learning / Deep Learning (Part 1) - Feedforward neural networks (FNN) / Hands-on: Deep Learning (Part 1) - Feedforward neural networks (FNN). Galaxy Training Network. Recuperado 19 de octubre de 2024, de

<https://training.galaxyproject.org/training-material/topics/statistics/tutorials/FNN/tutorial.html>

LinkedIn. (n.d.). (2) Modelo de regresión lineal basado en perceptrones | LinkedIn. <https://www.linkedin.com/pulse/perceptron-based-linear-regression-model-hemant-thapa-dpg3e/>

McGonagle, J., García, J. A., & Mollick, S. (n.d.). Feedforward Neural Networks | Brilliant Math & Science Wiki. Recuperado 19 de octubre de 2024, de <https://brilliant.org/wiki/feedforward-neural-networks/>

Rasamoelina, A. D., Adjailia, F., & Sincak, P. (2020). A Review of Activation Function for Artificial Neural Network. IEEE.

<https://doi.org/10.1109/sami48414.2020.9108717>