Aprendizaje Profundo (2024-2025)
Máster en Robótica e Inteligencia Artificial
Universidad de León

Práctica 3

Sheila Martínez Gómez
Alejandro Mayorga Caro
Ángel Morales Romero

7 de diciembre de 2024

Índice

1 Abstract **3**

2 Estudio del dataset **3**

3 Clasificación **4**

3.1 Preprocesamiento 4
3.2 Creación de la red neuronal 5
3.3 Entrenamiento de la red neuronal 6
3.4 Estudio capa a capa 7
3.4.1 Input Layer (`input_layer`) 8
3.4.2 Primera Capa Densa (`layer1`) 8
3.4.3 Segunda Capa Densa (`layer2`) 8
3.4.4 Tercera Capa Densa (`layer3`) 9
3.4.5 Capa de Salida (`output_layer`) 9
3.4.6 Relación con los Gráficos 10

4 Conclusiones **10**

1. Abstract

En este proyecto se va a realizar la siguiente tarea:
Uso de un conjunto de datos difícilmente separables para entrenar una red MLP multicapa con el objetivo de realizar la clasificación de dichos datos.
Una vez entrenada se visualizarán los datos a la salida de cada una de las capas, aportando la interpretación de la transformación efectuada por dicha capa.
El objetivo es reconocer las distintas transformaciones que las capas van aplicando a los datos de partida hasta convertirlos en linealmente separables.

2. Estudio del dataset

El dataset escogido es el *Moons Dataset*, que se obtiene de **Sklearn**. Es un dataset difícilmente separable, que contiene datos que forman dos semicírculos o lunas que no se superponen. Es un dataset utilizado normalmente para visualizar *clustering* y trabajar con algoritmos de clasificación.
En este caso se ha inicializado con los siguientes valores:

- **n_samples**: 2000
- **noise**: 0.25
- **random_state**: 0

Si bien es importante recalcar que el valor **random_state** no tiene ningún efecto muy significativo, pues únicamente sirve para asegurar la replicabilidad de las pruebas que se realizan.

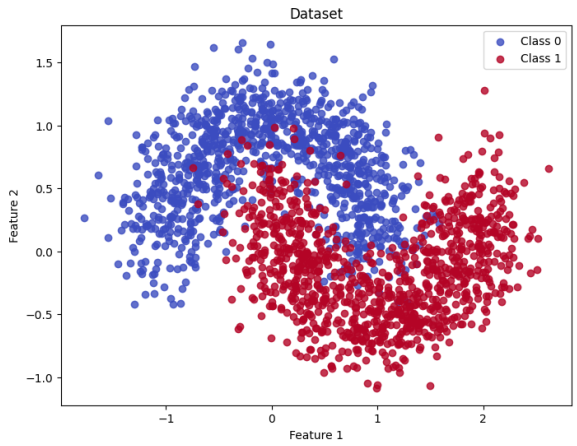

Figura 2.1: Representación del conjunto *moons dataset*

Como puede verse, este conjunto no puede separarse de manera precisa mediante un clasificador lineal, por lo que cumple a la perfección los requisitos de la tarea.
Consta de dos clases, que llamaremos **Clase 0** y **Clase 1**.

```
import numpy as np
import matplotlib.pyplot as plt
```

```
from sklearn.datasets import make_moons

# Number of samples
n_samples = 2000

# Generate the dataset
X, y = make_moons(n_samples=n_samples, noise=0.25, random_state=0)
```

3. Clasificación

3.1. Preprocesamiento

Para comenzar a trabajar con este dataset se debe realizar una división del conjunto en tres grupos de datos, **entrenamiento**, **validación** y **test**. Se ha escogido la siguiente distribución:

- **Entrenamiento**: 64 % del conjunto.
- **Validación**: 16 % del conjunto.
- **Test**: 20 % del conjunto.

```
xtrn, xtst, ytrn, ytst = train_test_split(X, y, test_size=0.2, random_state=seed)
xtrn, xval, ytrn, yval = train_test_split(xtrn, ytrn, test_size=0.2, random_state=seed)
```

Si imprimimos el argumento *shape* de cada set, podremos confirmar que el conjunto se ha separado correctamente:

```
Training set shape: (1280, 2) (1280,)
Validation set shape: (320, 2) (320,)
Test set shape: (400, 2) (400,)
```

Si mostramos uno de los sets, podremos confirmar que los datos que se han extraído siguen estando generalizados con respecto al conjunto inicial:

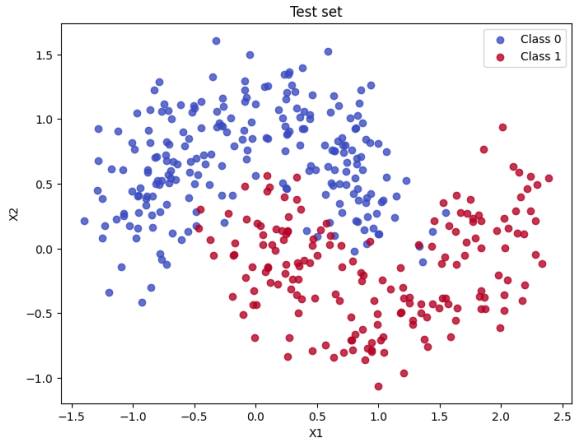

Figura 3.1: Representación del conjunto de test generado

3.2. Creación de la red neuronal

Para realizar la división del conjunto se utilizará una red neuronal con modelo **secuencial**. Constará de las capas propuestas a continuación:

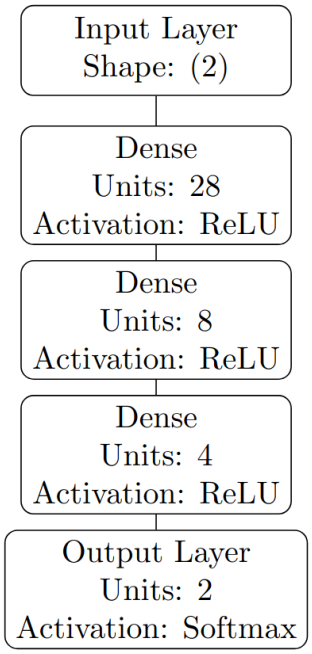

Figura 3.2: Representación de la red neuronal escogida

También se utilizará un valor de ***learning rate*** de 0.001 para conseguir que el modelo aprenda de manera estable y que el modelo converja más rápidamente.

```
# Create the MLP
model = Sequential()

input_layer = Input(shape=(2,), name='input_layer')
layer1 = Dense(28, activation='relu', name='layer1')(input_layer)
layer2 = Dense(8, activation='relu', name='layer2')(layer1)
layer3 = Dense(4, activation='relu', name='layer3')(layer2)
output_layer = Dense(2, activation='softmax', name='output_layer')(layer3)

model = Model(inputs=input_layer, outputs=output_layer)

# Compile the model
optimizer = Adam(learning_rate=0.001)
model.compile(optimizer=optimizer, loss='sparse_categorical_crossentropy', metrics=['accura
```

Si imprimimos el resumen del modelo obtendremos el número de parámetros y la forma que tiene cada una de las capas:

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 2)	0
layer1 (Dense)	(None, 28)	84
layer2 (Dense)	(None, 8)	232
layer3 (Dense)	(None, 4)	36
output_layer (Dense)	(None, 2)	10

Tabla 3.1: Resumen de la red neuronal

3.3. Entrenamiento de la red neuronal

Para entrenar la red neuronal y adaptarse al hardware utilizado, se dividirá el conjunto para obtener un *batch-size* de cinco grupos.
Posteriormente, se entrenará el modelo durante 220 épocas.

```
# Train the model
bs = xtrn.shape[0]//5
history = model.fit(xtrn, ytrn, epochs=220, batch_size=bs, validation_data=(xval, yval))
```

Una vez entrenado, y para obtener los valores que realmente son relevantes, evaluamos el modelo en el conjunto de test. Esto nos permitirá comprobar si la red generaliza bien, si se ajusta demasiado a los datos (*overfitting*), o si no es lo suficientemente potente y no se ajusta lo necesario (*underfitting*).

```
test_loss, test_accuracy = model.evaluate(xtst, ytst)  # xtst y ytst son los datos de p
```

Si imprimimos las curvas de progreso de *accuracy* y *loss*, vemos que el modelo está bien ajustado, que converge correctamente, y que no sufre de *underfitting* u *overfitting*.

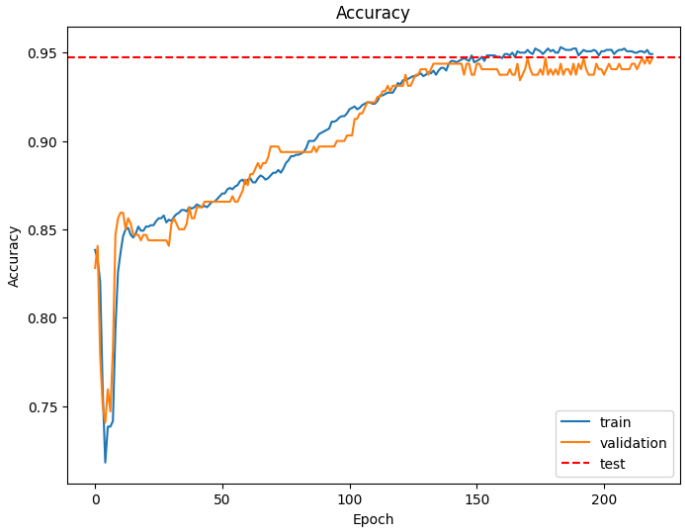

Figura 3.3: *Accuracy* en el set de test

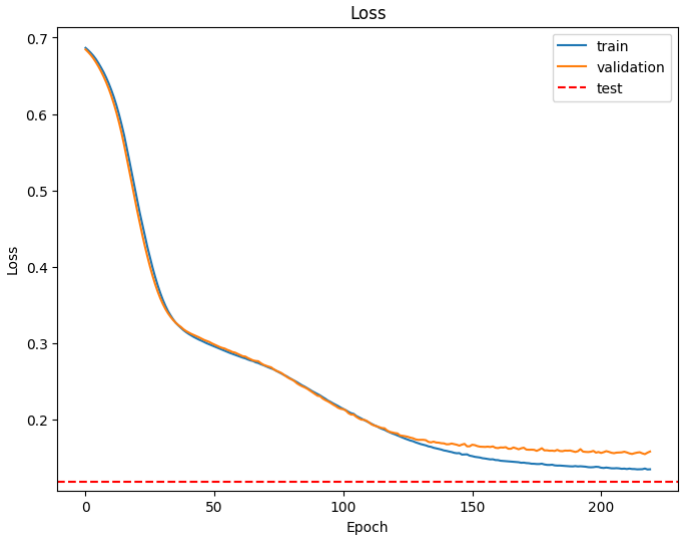

Figura 3.4: Pérdida en el set de test

Además, el modelo obtiene una precisión de 0.9495, un valor que puede considerarse válido en esta situación.

3.4. Estudio capa a capa

En esta sección se tratarán las diferentes transformaciones que aplican las neuronas en cada capa del modelo para separar los dos conjuntos del set.
La red neuronal analizada tiene cinco capas que cumplen funciones específicas y transforman los datos para lograr una representación abstracta que facilite la clasificación.

3.4.1. Input Layer (`input_layer`)

La capa de entrada tiene una forma de `(None, 2)`, lo que significa que recibe datos de entrada con dos características (*Feature 1* y *Feature 2*). Esta capa no realiza transformaciones y simplemente pasa los datos a la siguiente capa.

3.4.2. Primera Capa Densa (`layer1`)

La primera capa densa contiene 28 neuronas y utiliza la función de activación ReLU ($\text{ReLU}(x) = \text{máx}(0, x)$). Su propósito es aprender combinaciones lineales de las características de entrada y agregar no linealidad al modelo. Esto amplía el espacio de representación de las características iniciales a 28 dimensiones, permitiendo al modelo detectar patrones más complejos.

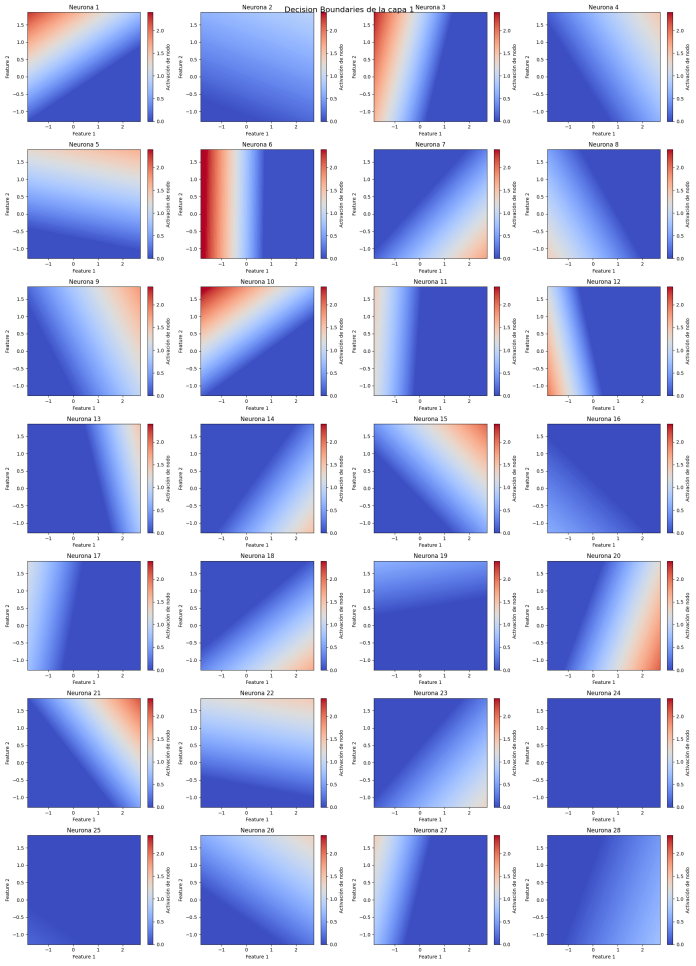

Figura 3.5: Límites de decisión de las neuronas de la primera capa densa del modelo

3.4.3. Segunda Capa Densa (`layer2`)

La segunda capa densa tiene 8 neuronas y también utiliza la función de activación ReLU. Esta capa reduce la dimensionalidad de las activaciones previas, pasando de 28 a 8 dimensiones. Las gráficas asociadas muestran cómo las activaciones de estas neuronas responden en función de las dos características de entrada. En los gráficos, las zonas rojas representan

valores de activación altos, mientras que las zonas azules representan valores bajos o nulos, resultado del efecto de la función ReLU. Cada neurona en esta capa detecta patrones específicos en los datos de entrada. Algunas neuronas, como la Neurona 5, muestran activaciones planas, lo que podría indicar que tienen menor relevancia en el modelo.

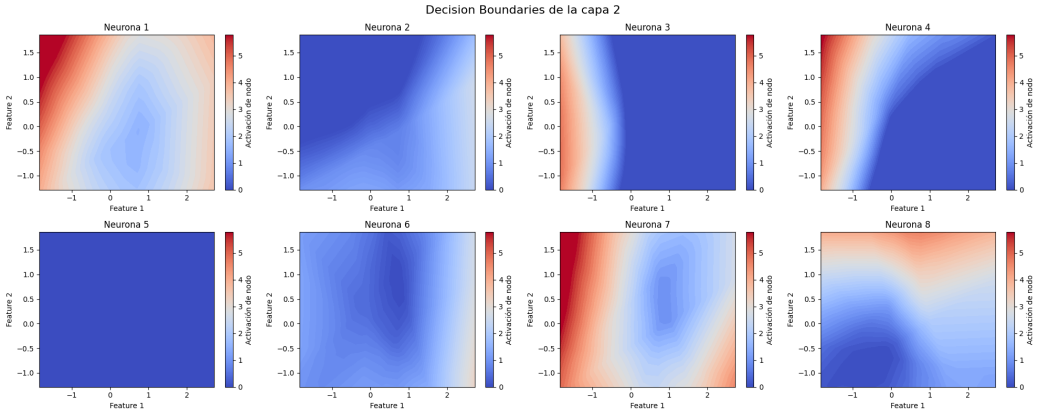

Figura 3.6: Límites de decisión de las neuronas de la segunda capa densa del modelo

3.4.4. Tercera Capa Densa (`layer3`)

La tercera capa densa tiene 4 neuronas y utiliza la función de activación ReLU. Su objetivo es reducir aún más la dimensionalidad, de 8 a 4, mientras detecta patrones más abstractos basados en las activaciones de la capa anterior. La introducción de no linealidades en esta etapa permite al modelo crear representaciones más condensadas y jerárquicas de los datos.

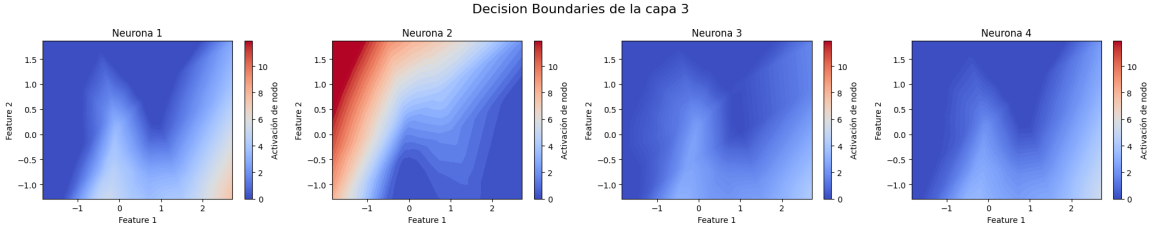

Figura 3.7: Límites de decisión de las neuronas de la tercera capa densa del modelo

3.4.5. Capa de Salida (`output_layer`)

La capa de salida tiene 2 neuronas y utiliza la función de activación Softmax. Esta capa genera dos valores de salida que representan las probabilidades de pertenecer a dos clases diferentes (por ejemplo, Clase A y Clase B). La función Softmax se define como

$$\text{Softmax}(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}},$$

lo que asegura que las salidas estén normalizadas, es decir, que la suma de las probabilidades sea igual a 1. Esta capa es utilizada para realizar la clasificación final.

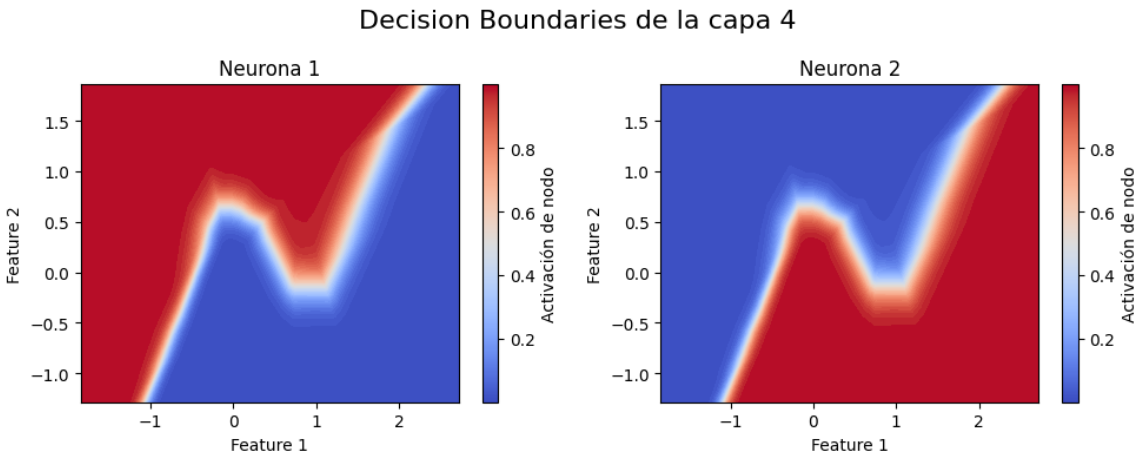

Figura 3.8: Límites de decisión de las neuronas de la cuarta capa densa del modelo

3.4.6. Relación con los Gráficos

Los gráficos que se presentan corresponden a los límites de decisión (*decision boundaries*) aprendidos por las 8 neuronas de la segunda capa densa (`layer2`).
Cada gráfico representa la activación de una neurona específica en función de las dos características de entrada (*Feature 1* y *Feature 2*).
Las zonas rojas indican activaciones altas, mientras que las zonas azules indican activaciones bajas o nulas. Neuronas con límites claros entre zonas (como las Neuronas 1, 3 y 4) están captando patrones relevantes en los datos. Por otro lado, neuronas como la Neurona 5, con una activación uniforme, podrían tener menor relevancia para el modelo.

4. Conclusiones

En esta práctica se ha llevado a cabo el entrenamiento de una red MPL multicapa con el fin de clasificar los datos de un conjunto difícilmente separable, en este caso utilizando el dataset *Moons Dataset*. A lo largo del proyecto, se abordaron las siguientes etapas clave: análisis del dataset, preprocesamiento, diseño de la red neuronal, entrenamiento y un estudio detallado de las transformaciones realizadas en cada capa de la red.
Tras diseñar la red como aparece en la figura ?? y entrenarla con el dataset (64 % Entrenamiento, 16 % validación y 20 % test), obtenemos una precisión del 94.95 %, lo cual muestra un desempeño notable del modelo. Esto evidencia que el modelo fue capaz de generalizar adecuadamente y de evitar problemas de overfitting o underfitting, demostrando estabilidad durante el proceso de aprendizaje.
Después de obtener este resultado, se han analizado los resultados generados capa por capa hasta llegar al resultado final y esto ha permitido comprender cómo la red ha sido capaz de transformar el espacio de entrada inicial en representaciones abstractas pero linealmente separables.
En resumen, este proyecto ha demostrado el potencial de las redes MPL multicapa para resolver problemas de clasificación difícilmente separables, analizando también los resultados obtenidos capa a capa.

Apéndice

El código relacionado con este documento está disponible en el siguiente enlace de GitHub:

github.com/angelleon01/aprendizaje_profundo