INSTITUTO TECNOLÓGICO DE COSTA RICA

ÁREA ACADÉMICA DE INGENIERÍA MECATRÓNICA

MT 8008
Inteligencia Artificial

Tarea 1 – Parte 1

Sistemas Conexionistas

Jose Fabio Navarro Naranjo – 2019049626

César Argüello Salas – 2019047699

Profesor: Juan Luis Crespo Mariño

Semestre II - 2022

Parte 1

1- *Estudio de hiperparámetros:*

Lista de hiperparámetros a analizar:

- Taza de aprendizaje
- Número de capas
- Número de neuronas
- Función de activación
- Optimizador
- Función de Pérdida
- Número de iteraciones

a. Clasificación de muestras de leche

Las iteraciones van a comenzar con los siguientes hiperparámetros:

- Taza de aprendizaje = 0.0001
- Número de capas ocultas = 1
- Número de neuronas = 5
- Función de activación = sigmoide
- Optimizador = adam
- Función de Pérdida = crossentropy
- Número de iteraciones = 30

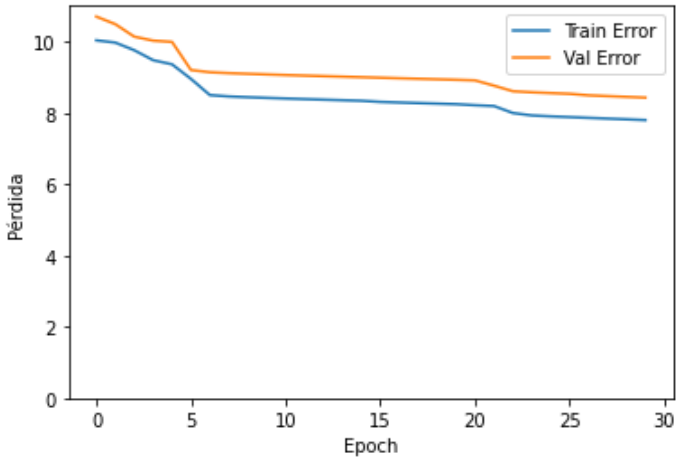

Figura 1. Iteración 1 para la red de clasificación de muestras de leche.

Como se puede observar, si existe una disminución en el valor del error conforme aumentan las iteraciones. En este caso, aproximadamente luego de la iteración número 6, el error se estanca y deja de disminuir, sin embargo, luego de la iteración 18 aproximadamente vuelve a disminuir. Por lo que resulta interesante probar con una mayor cantidad de iteraciones.

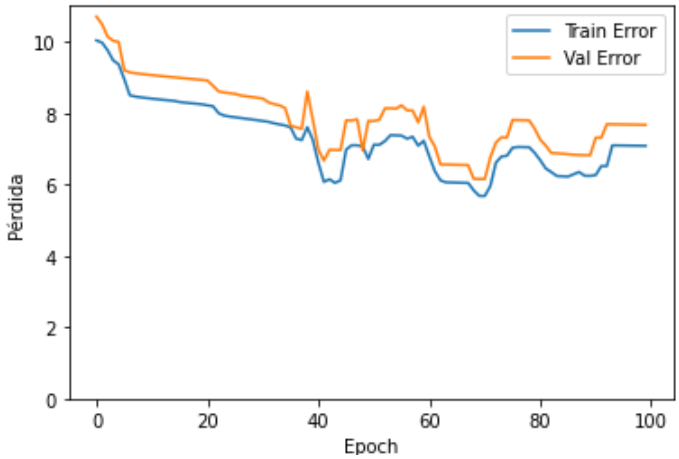

Figura 2. Iteración 2 para la red de clasificación de muestras de leche.

Es importante mencionar que tanto la función de pérdida, como la función de activación si bien, se pueden variar, para los efectos de esta red, no se variarán, debido a que son las funciones recomendadas.

Al analizar el resultado de la iteración anterior, se puede notar que la pérdida no sigue un comportamiento uniforme, y que inclusive tiene tendencias de aumento. Debido a esto, se decide probar con una mayor cantidad de neuronas, para conocer si el comportamiento se estabiliza.

- Taza de aprendizaje = 0.0001
- Número de capas ocultas = 1
- Número de neuronas = 7
- Función de activación = sigmoide
- Optimizador = adam
- Función de Pérdida = crossentropy
- Número de iteraciones = 100

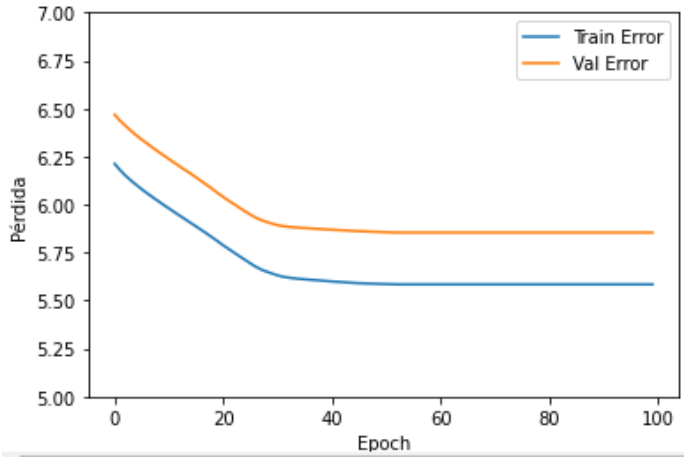

Figura 3. Iteración 3 para la red de clasificación de muestras de leche.

Ahora, se observa que aproximadamente luego 40 iteraciones se estabiliza la pérdida. Además, se consigue una curva de aprendizaje con una mejor forma, obteniendo una pérdida final de 5.58.

A continuación, se realizará otra iteración adicional, variando el número de neuronas de la capa oculta, para verificar si se puede obtener una menor pérdida.

- Taza de aprendizaje = 0.0001
- Número de capas ocultas = 1
- Número de neuronas = 9
- Función de activación = sigmoide
- Optimizador = adam
- Función de Pérdida = crossentropy
- Número de iteraciones = 100

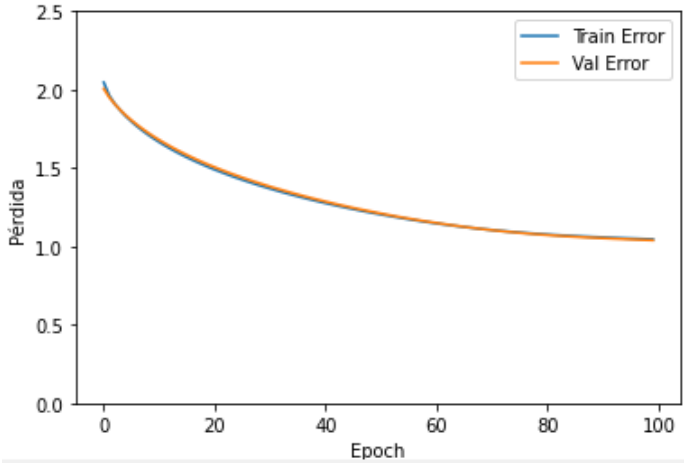

Figura 4. Iteración 4 para la red de clasificación de muestras de leche.

En este caso, se observa una línea de aprendizaje satisfactoria, ya que además de ser tener un comportamiento deseado, tiene una pérdida muchísimo menor que en el caso anterior.

Para el parámetro de la cantidad de neuronas por capa, se probó con números mayores y menores a 9, y la pérdida se dispara, por lo cual se para esta combinación de hiperparámetros, 9 neuronas en la capa oculta son el valor ideal.

De igual manera, se continuó iterando, variando el hiperparámetro de la cantidad de capas, y para números mayores a 1, la pérdida se dispara a números mucho mayores.

Ahora bien, se mantuvo la cantidad de 9 neuronas, pero se aumentó 10 veces la tasa de aprendizaje, para conocer las repercusiones de este cambio en el modelo, por lo que se utilizaron los siguiente hiperparámetros:

- Taza de aprendizaje = 0.001
- Número de capas ocultas = 1
- Número de neuronas = 9

- Función de activación = sigmoide
- Optimizador = adam
- Función de Pérdida = crossentropy
- Número de iteraciones = 100

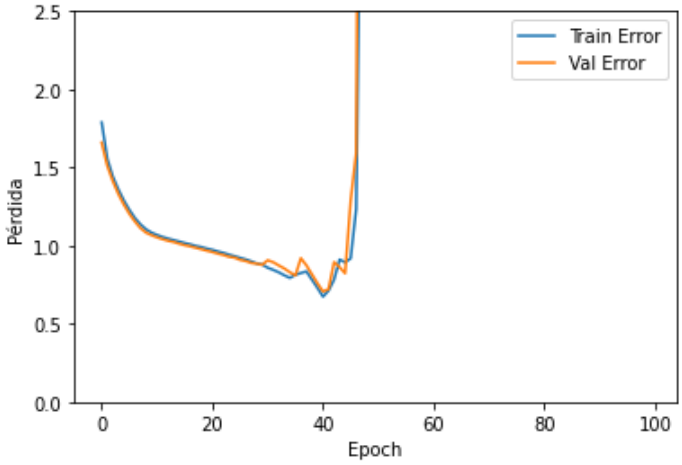

Figura 5. Iteración 5 para la red de clasificación de muestras de leche.

Se puede notar que el comportamiento obtenido no fue el deseado.

Además, se realizó una iteración con una disminución de 10 veces en la tasa de aprendizaje, cuyo resultado fue el que se muestra a continuación.

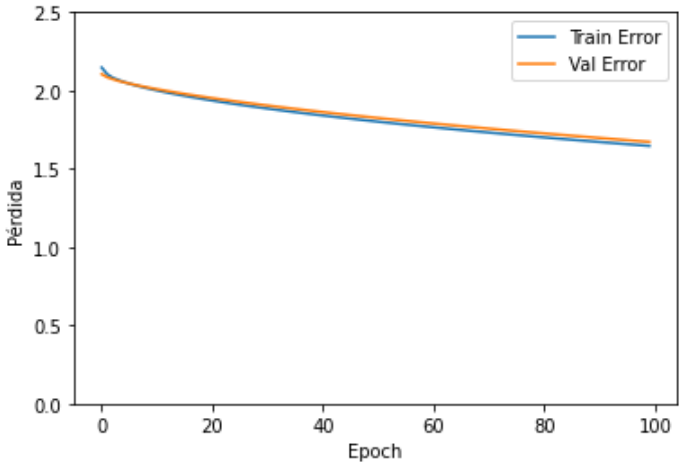

Figura 6. Iteración 6 para la red de clasificación de muestras de leche.

Se puede observar que, aunque la curva desciende, no alcanza una pérdida tan pequeña como con la tasa de aprendizaje de 0.0001. Entonces, para esta combinación, la tasa de aprendizaje óptima es 0.0001.

Por último, se van a probar distintas funciones optimizadoras para la combinación actual. Para esta iteración, se va a utilizar el optimizador SGD, que es un descenso de gradiente, con optimizador de momento.

- Taza de aprendizaje = 0.0001
- Número de capas ocultas = 1
- Número de neuronas = 9
- Función de activación = sigmoide
- Optimizador = SGD
- Función de Pérdida = crossentropy
- Número de iteraciones = 100

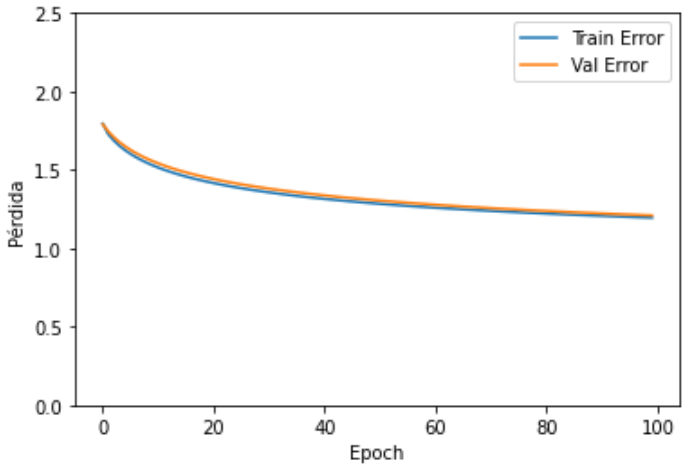

Figura 6. Iteración 6 para la red de clasificación de muestras de leche.

En este caso, se observa que existe aprendizaje, y que se alcanza una pérdida similar a la pérdida obtenida con la función ADAM.

Para el caso de la función que implementa el algoritmo Adadelta, que se muestra en la siguiente figura, no se obtiene aprendizaje del todo.

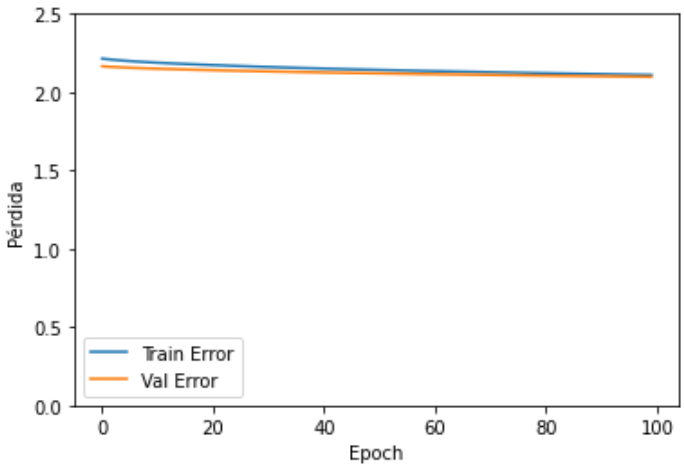

Figura 7. Iteración 7 para la red de clasificación de muestras de leche.

De igual manera, se probó con algoritmos como Adagrad, y Adamax, y no se obtuvieron resultados relevantes.

Debido a lo anterior, se concluye que la combinación óptima para el caso de la clasificación de muestras de leche es:

- Taza de aprendizaje = 0.0001
- Número de capas ocultas = 1
- Número de neuronas = 9
- Función de activación = sigmoide
- Optimizador = adam
- Función de Pérdida = crossentropy
- Número de iteraciones = 100

b. **Estimación de calidad de vino blanco**

Las iteraciones van a comenzar con los siguientes hiperparámetros:

- Taza de aprendizaje = 0.0001
- Número de capas ocultas = 1
- Número de neuronas = 11
- Función de activación = sigmoide
- Optimizador = adam
- Función de Pérdida = mse
- Número de iteraciones = 30

Para este caso, se obtuvo la siguiente curva de aprendizaje.

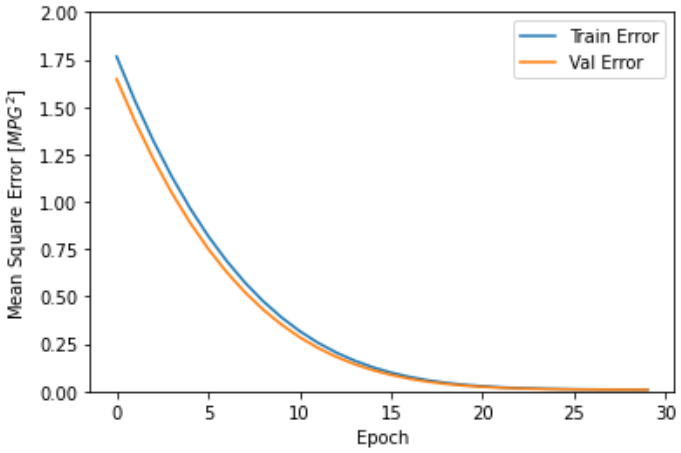

Figura 8. Iteración 1 para la red de estimación de calidad de vino blanco.

Se puede observar que la red tiene un gran aprendizaje, y al cabo de poco más de 20 iteraciones, aproxima el error a 0. En este caso, se van a realizar más pruebas con el fin de conocer si con una menor de iteraciones por prueba, se puede aproximar la pérdida a 0 como se hace en este caso.

Luego de varias pruebas con un aumento y disminución, se obtuvo prácticamente el mismo comportamiento, sin embargo, al disminuir la cantidad de neuronas a 5, se obtuvo un punto óptimo, ya permite mantener una curva de aprendizaje con una pérdida baja, por lo que optimiza el comportamiento de la red. El resultado de dicha iteración se muestra en la siguiente gráfica.

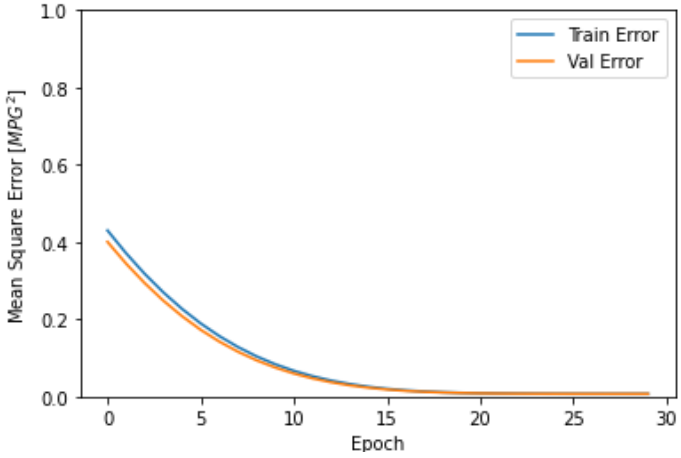

Figura 9. Iteración 2 para la red de estimación de calidad de vino blanco.

Ahora bien, se consideró el aumento de la cantidad de capas ocultas a 2, y con esto se obtuvo la siguiente gráfica de pérdida.

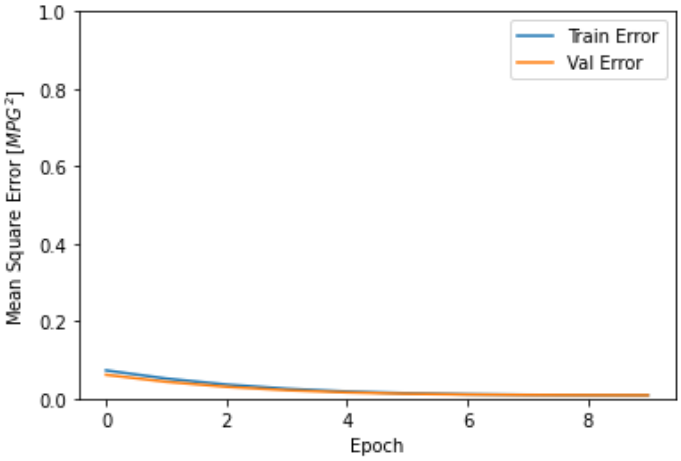

Figura 10. Iteración 3 para la red de estimación de calidad de vino blanco.

A pesar de que, en la gráfica anterior, hay una pérdida muy baja con pocas iteraciones, se observa que el aprendizaje de la red es bastante pequeño, por lo cual es preferible continuar trabajando con solo una capa.

Ahora, se va a probar modificando la tasa de aprendizaje, por lo que la siguiente prueba, va a ser corrida bajo los siguientes hiperparámetros:

- Taza de aprendizaje = 0.001

- Número de capas ocultas = 1
- Número de neuronas = 5
- Función de activación = sigmoide
- Optimizador = adam
- Función de Pérdida = mse
- Número de iteraciones = 30

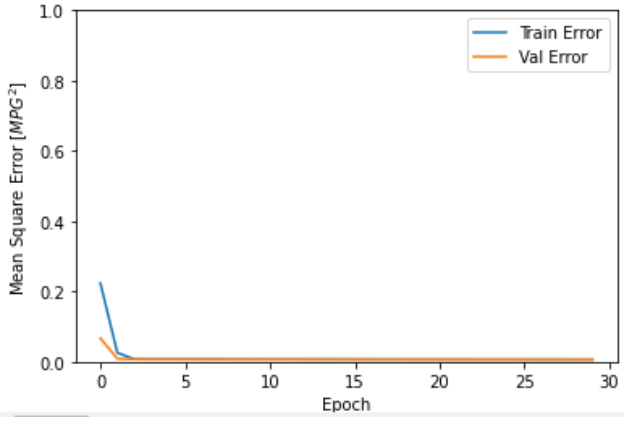

Figura 11. Iteración 4 para la red de estimación de calidad de vino blanco.

Y ahora, se decide disminuir la tasa de aprendizaje. Se aumentan las iteraciones para poder observar mejor el comportamiento de la red.

- Taza de aprendizaje = 0.00001
- Número de capas ocultas = 1
- Número de neuronas = 5
- Función de activación = sigmoide
- Optimizador = adam
- Función de Pérdida = mse
- Número de iteraciones = 130

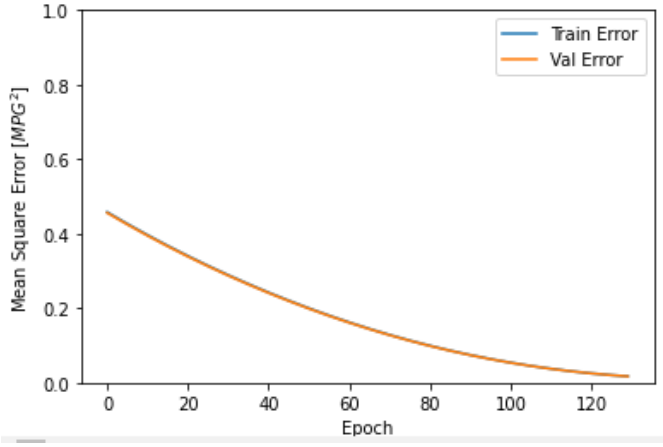

Figura 12. Iteración 5 para la red de estimación de calidad de vino blanco.

En las figuras 11 y 12, se puede observar respectivamente, un aumento y una disminución de la tasa de aprendizaje en un factor 10. Para el caso del aumento, se puede ver que el periodo de aprendizaje de la red es muy corto, y abrupto, si embargo para el caso del aumento, se ve un proceso más lento y continuo, pero con una mayor cantidad de iteraciones.

Debido a los 2 casos anteriores, se decide continuar con la tasa de aprendizaje original de 0.0001.

Ahora bien, se va a probar la función de pérdida de suma absoluta del error (mae) bajo los siguientes hiperparámetros, para verificar si se puede obtener un mejor desempeño en la red.

- Taza de aprendizaje = 0.0001
- Número de capas ocultas = 1
- Número de neuronas = 5
- Función de activación = sigmoide
- Optimizador = adam
- Función de Pérdida = mae
- Número de iteraciones = 30

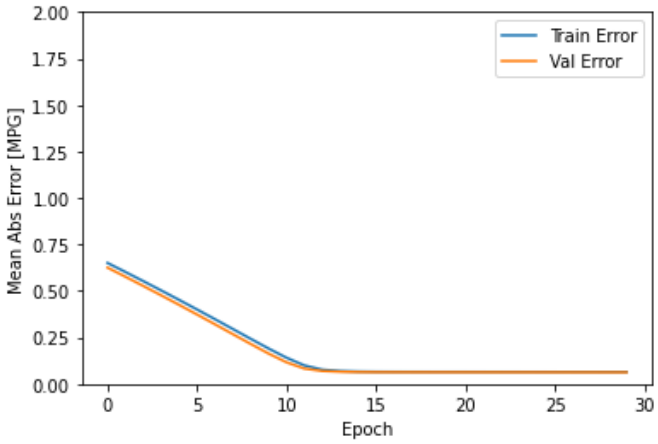

Figura 13. Iteración 6 para la red de estimación de calidad de vino blanco.

En este caso se observa que la caída en el error es prácticamente lineal hasta que se estanca luego de 12 iteraciones aproximadamente en 0.06. Sin embargo, si se compara este comportamiento con el obtenido en la figura 9, ya que, para este caso, con la función de pérdida de suma cuadrática del error, la pérdida se estabiliza en un valor menor. Debido a esto, se continúa trabajando con la función de suma cuadrática del error para la pérdida (mse).

Por último, se van a realizar pruebas con diversas función de optimización. Primero, se va a probar la función del algoritmo SGD, con la siguiente configuración.

- Taza de aprendizaje = 0.0001
- Número de capas ocultas = 1
- Número de neuronas = 5
- Función de activación = sigmoide

- Optimizador = SGD
- Función de Pérdida = mse
- Número de iteraciones = 30

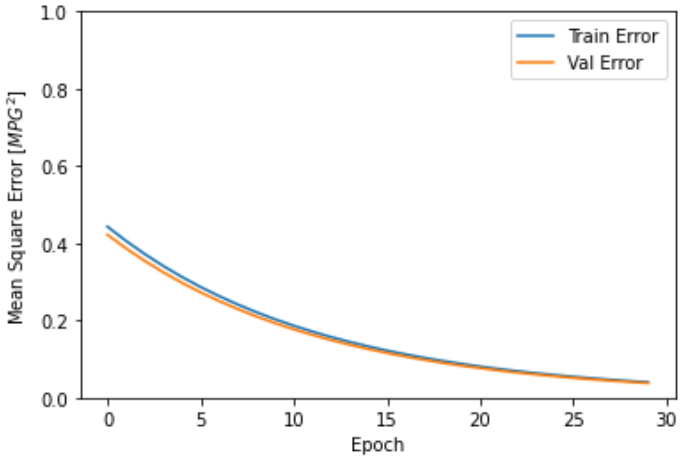

Figura 14. Iteración 7 para la red de estimación de calidad de vino blanco.

Observando el comportamiento obtenido en la figura 14, para la función de optimización SGD, se puede notar que si bien, existe una curva de aprendizaje satisfactoria, este aprendizaje se da con una mayor cantidad de iteraciones, a diferencia del caso en el que se utilizó la función de pérdida “Adam”.

Seguidamente, se prueba la misma configuración con el optimizador “Adadelta”, sin embargo, no se muestra aprendizaje del todo, ya que el error nunca disminuye. Lo mismo sucede con el optimizador “Adagrad”. Sin embargo, para el caso del optimizador “Adamax”, se obtiene un resultado similar al obtenido con el optimizador “Adam”, el cual se muestra en la figura 15.

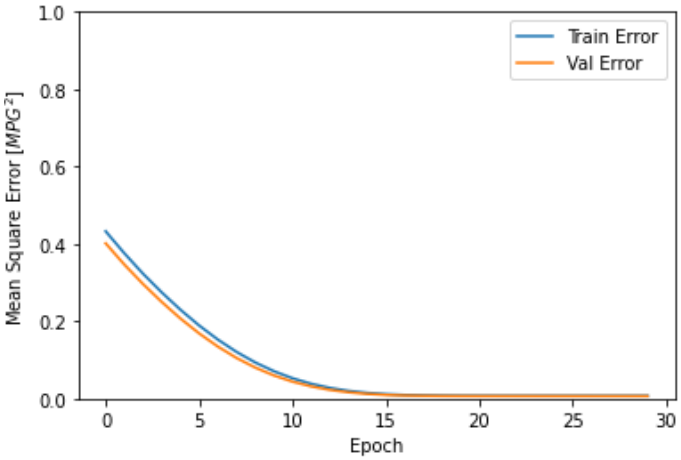

Figura 15. Iteración 8 para la red de estimación de calidad de vino blanco.

Como se ya se mencionó, en este caso se obtuvo un comportamiento similar al obtenido en la figura 9, sin embargo, se prefiere utilizar el optimizador “Adam” ya que es el recomendado.

Por lo que, finalmente, estos son los hiperparámetros definidos para esta red:

- Taza de aprendizaje = 0.0001
- Número de capas ocultas = 1
- Número de neuronas = 5
- Función de activación = sigmoide
- Optimizador = Adam
- Función de Pérdida = mse
- Número de iteraciones = 20

c. **Estimación de calidad de vino tinto**

Para este caso, se comienza el estudio con los siguientes parámetros:

- Taza de aprendizaje = 0.0001
- Número de capas ocultas = 1
- Número de neuronas = 11
- Función de activación = sigmoide
- Optimizador = Adam
- Función de Pérdida = mse
- Número de iteraciones = 80

Con lo anterior, se obtiene el comportamiento mostrado en la figura 16, para la curva de aprendizaje.

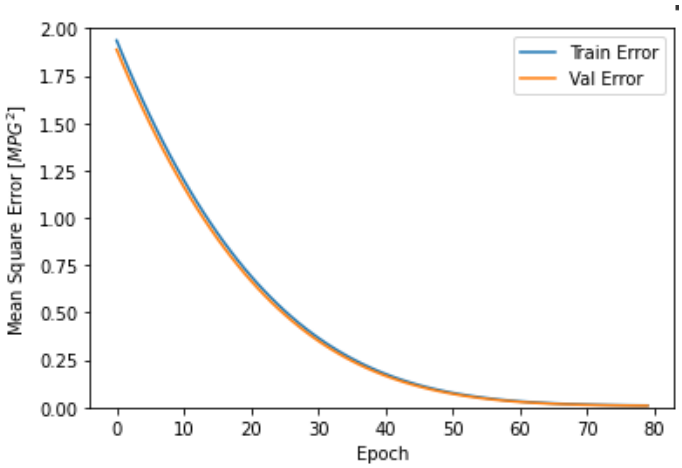

Figura 16. Iteración 1 para la red de estimación de calidad de vino tinto.

Para este caso se ve un buen comportamiento, sin embargo, hay una gran cantidad de neuronas en la capa oculta, por lo que se van a reducir estas hasta un punto donde se mantenga el comportamiento observado.

Luego de algunas iteraciones, se llegó a la cantidad de 5 neuronas en la capa oculta, con lo cual se obtuvo un aprendizaje satisfactorio, y una reducción bastante grande de la pérdida (hasta 0.0055 aproximadamente). Dicho resultado se muestra en la figura 17.

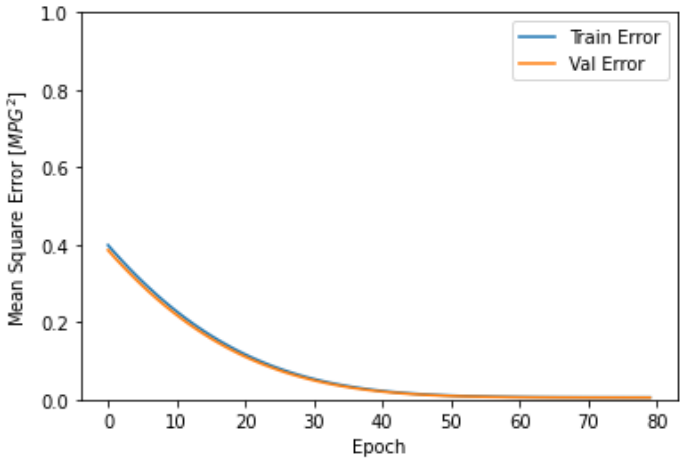

Figura 17. Iteración 2 para la red de estimación de calidad de vino tinto.

Ahora bien, respecto a la tasa de aprendizaje, actualmente se encuentra en 0.0001. Seguidamente se va a probar con un aumento, y una disminución de 10 veces en ambos casos.

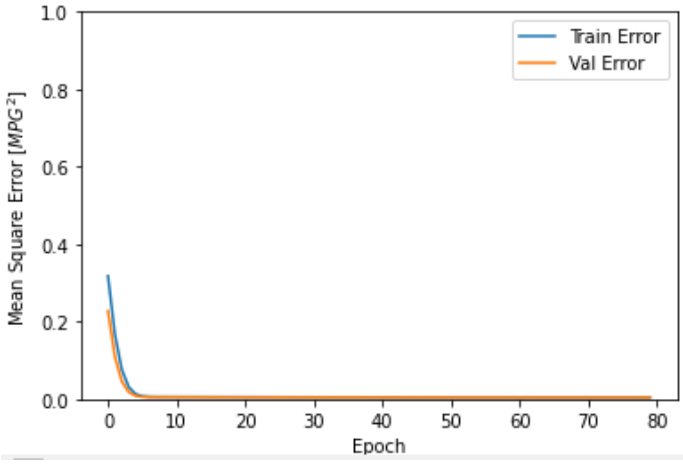

Figura 18. Iteración 3 para la red de estimación de calidad de vino tinto.

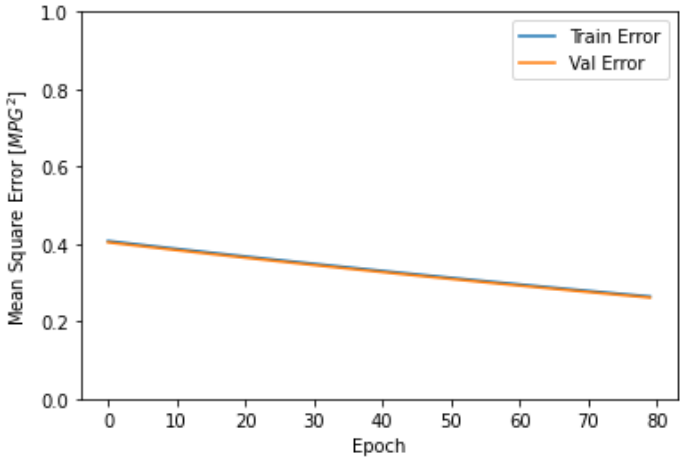

Figura 19. Iteración 4 para la red de estimación de calidad de vino tinto.

Primero, en la figura 18, para el caso del aumento de la tasa de aprendizaje, se observa un aprendizaje bastante rápido, que de igual manera obtiene una pérdida bastante baja, en una cantidad menor de iteraciones, lo que resulta satisfactorio.

Por el otro lado, en la figura 19, con una tasa de aprendizaje mucho menor se observa que con 80 iteraciones la red ni siquiera ha podido estabilizar la pérdida, lo cual indica que se van a necesitar muchas más iteraciones y, por ende, mayor tiempo de procesamiento.

Debido a lo anterior, se decide seguir con un aprendizaje de 0.001, con una cantidad aproximada de 10 iteraciones.

Ahora bien, se va a probar un aumento en las capas de la red. Por lo que se va a realizar una iteración con los siguientes parámetros:

- Taza de aprendizaje = 0.001
- Número de capas ocultas = 2
- Número de neuronas = 5
- Función de activación = sigmoide
- Optimizador = Adam
- Función de Pérdida = mse
- Número de iteraciones = 10

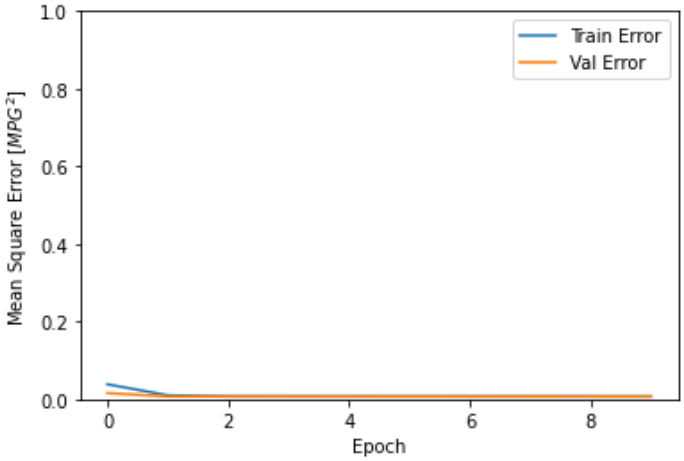

Figura 20. Iteración 5 para la red de estimación de calidad de vino tinto.

Para este caso, en la figura 20 se observa el resultado con 2 capas ocultas, y a pesar de que se obtiene una pérdida bastante baja en la red, se nota que el aprendizaje es bastante pequeño, por lo cual se prefiere continuar con una única capa oculta.

Ahora bien, se va a probar con la función de pérdida de suma de error absoluto (mae), para corroborar si se encuentra una mejora en la red. Los parámetros para dicha prueba se muestran a continuación:

- Taza de aprendizaje = 0.001
- Número de capas ocultas = 1
- Número de neuronas = 5
- Función de activación = sigmoide
- Optimizador = Adam
- Función de Pérdida = mae
- Número de iteraciones = 10

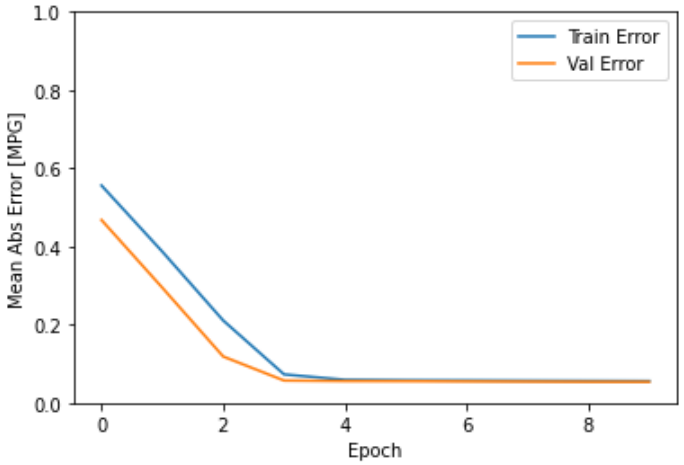

Figura 21. Iteración 6 para la red de estimación de calidad de vino tinto.

En la figura 21, se puede observar que, al utilizar como pérdida, la suma absoluta del error, se obtiene la curva de aprendizaje esperada, sin embargo, la pérdida se estanca en un valor mayor comparado al obtenido en la figura 18, para el caso de la función de pérdida de suma cuadrática del error. Debido a esto, se decide continuar con la pérdida propuesta originalmente.

Por último, se va a probar con algunos optimizadores distintos. Primero, se va a utilizar la siguiente configuración:

- Taza de aprendizaje = 0.001
- Número de capas ocultas = 1
- Número de neuronas = 5
- Función de activación = sigmoide
- Optimizador = SGD
- Función de Pérdida = mse
- Número de iteraciones = 10

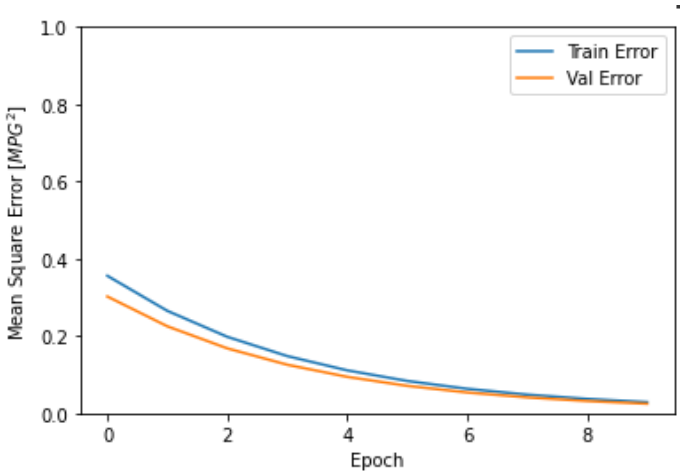

Figura 22. Iteración 7 para la red de estimación de calidad de vino tinto.

Para este caso, con el optimizador SGD, se obtuvo la gráfica de la figura 22, y se observa una curva de aprendizaje satisfactoria, sin embargo, con un descenso más lento que el obtenido mediante el optimizador “Adam”.

Ahora bien, con los optimizadores “Adadelta” y “Adagrad”, se obtienen aprendizajes prácticamente nulos para la cantidad de iteraciones realizadas, y debido a que el optimizador “Adam”, permite un aprendizaje mejor en este rango, no vale la pena probar con estos optimizadores en rangos mayores de iteraciones.

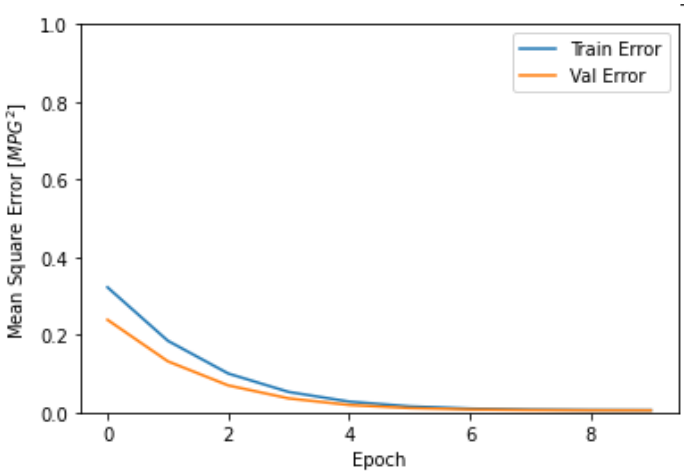

Figura 23. Iteración 8 para la red de estimación de calidad de vino tinto.

Por último, con el optimizador "Adamax" se obtiene la gráfica mostrada en la figura 23, donde se observa un aprendizaje bastante similar al obtenido con el optimizador "Adam", sin embargo, al igual que en el caso anterior, se prefiere continuar con el optimizador recomendado.

Debido al análisis anterior, los hiperparámetros definidos para la red de estimación de calidad de vino blanco son los siguientes:

- Taza de aprendizaje = 0.001
- Número de capas ocultas = 1
- Número de neuronas = 5
- Función de activación = sigmoide
- Optimizador = Adam
- Función de Pérdida = mse
- Número de iteraciones = 10

d. **Estimación de calidad de vino en general**

Para este caso, se comienza a probar con los siguientes hiperparámetros:

- Taza de aprendizaje = 0.0001
- Número de capas ocultas = 1
- Número de neuronas = 11
- Función de activación = sigmoide
- Optimizador = Adam
- Función de Pérdida = mse
- Número de iteraciones = 30

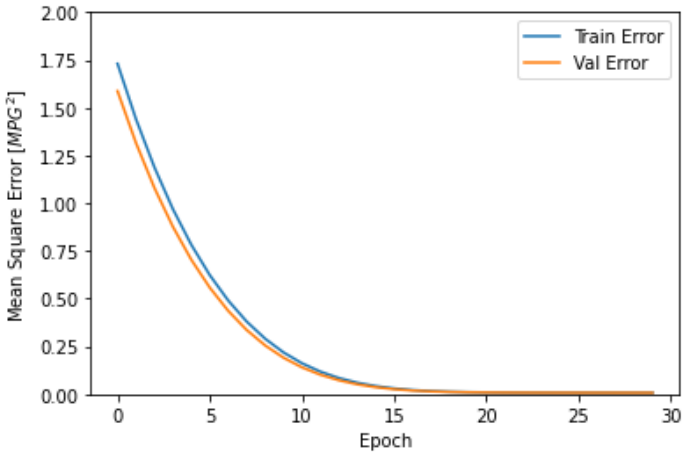

Figura 24. Iteración 1 para la red de estimación de calidad de vino en general.

En la figura 24, se puede observar que existe un comportamiento de aprendizaje satisfactorio, sin embargo, la cantidad de neuronas por capa es elevada, por lo cual, se realizaron iteraciones para obtener el menor número de neuronas que se pueden utilizar para obtener el mismo comportamiento que se observa en esta figura.

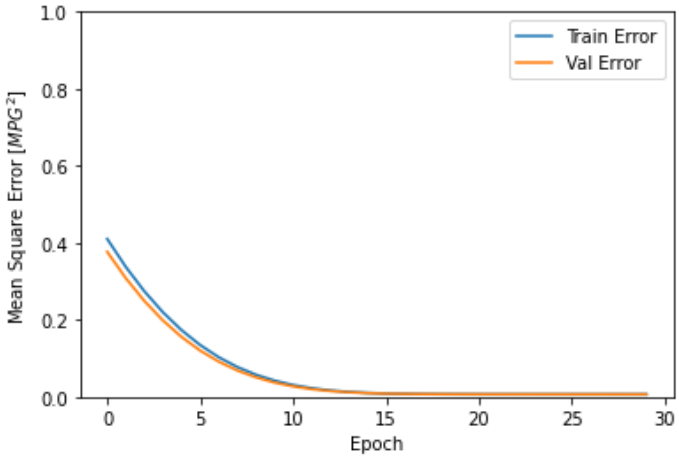

Figura 25. Iteración 2 para la red de estimación de calidad de vino en general.

Debido a lo anterior, se obtuvo que, con 5 neuronas en la capa oculta, se logran una pérdida bastante baja, con una curva de aprendizaje similar a la obtenida en la figura 24. Estos datos se muestran en la figura 25.

Ahora bien, respecto a la tasa de aprendizaje, al igual que en los casos anteriores, se va a realizar una prueba con un aumento y una disminución de 10 veces de la tasa propuesta originalmente.

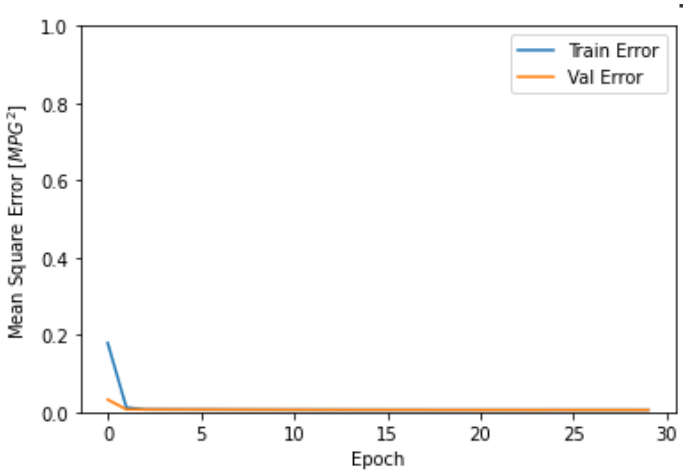

Figura 26. Iteración 3 para la red de estimación de calidad de vino en general.

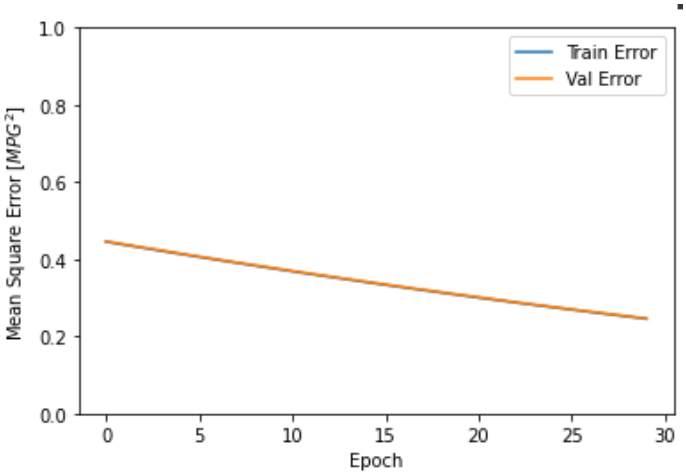

Figura 27. Iteración 4 para la red de estimación de calidad de vino en general.

En la figura 26, se muestra el resultado para una tasa de aprendizaje mayor, sin embargo, se nota un comportamiento muy abrupto, y con poco aprendizaje. Por el otro lado, para el caso donde se disminuye la tasa de aprendizaje, se obtiene un comportamiento con una caída excesivamente lenta, en comparación con el comportamiento original (con las condiciones propuestas). Debido a lo anterior, se va a continuar con la tasa de aprendizaje propuesta originalmente de 0.0001.

Ahora bien, se va a agregar un aumento en la cantidad de capas ocultas, utilizando los siguientes hiperparámetros, para verificar si existe alguna mejora sustancial en la red

- Taza de aprendizaje = 0.0001
- Número de capas ocultas = 2
- Número de neuronas = 5
- Función de activación = sigmoide
- Optimizador = Adam
- Función de Pérdida = mse

- Número de iteraciones = 30

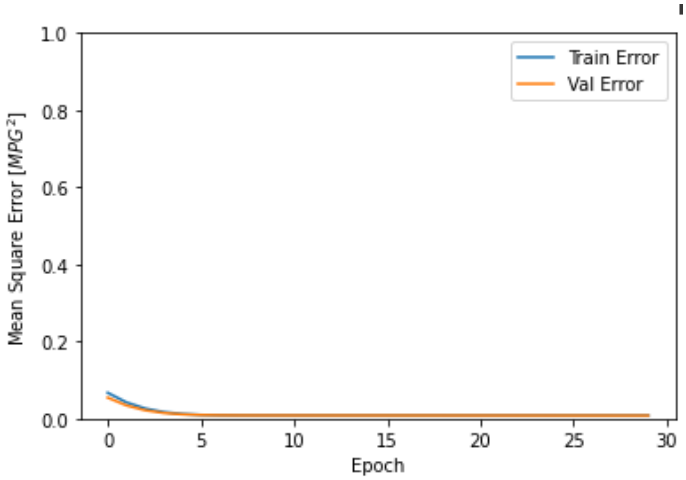

Figura 28. Iteración 5 para la red de estimación de calidad de vino en general.

Para el caso analizado en la figura 28, utilizando 2 capas ocultas, se puede observar que el grado de aprendizaje de la red es bastante bajo, por lo cual, es preferible continuar únicamente con una capa oculta.

Por otra parte, se realizó de igual manera, una prueba cambiando la función de pérdida. Esta ves utilizando la suma absoluta del error, en vez de la suma cuadrática.

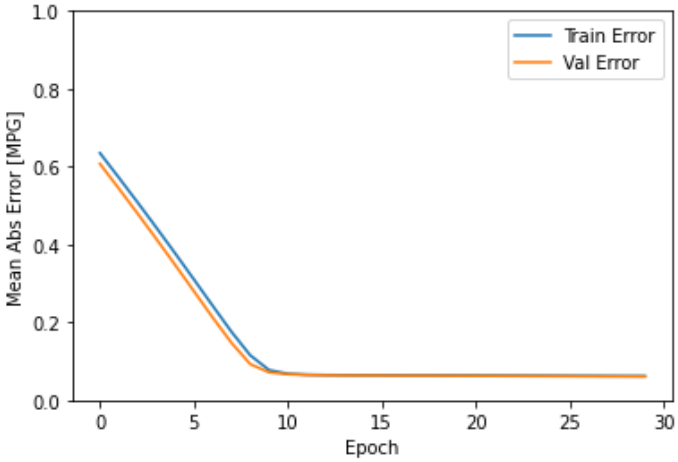

Figura 29. Iteración 6 para la red de estimación de calidad de vino en general.

Para este caso, en la figura 29, se observa una caída con tendencia lineal, hasta el punto de estancamiento de la pérdida. Dicha gráfica, a pesar de presentar un comportamiento de aprendizaje satisfactorio, el valor es que se estanca la pérdida es mayor que para el caso donde se utiliza la suma cuadrática del error, por lo cual se va a continuar utilizando este último mencionado.

Seguidamente, se van a realizar pruebas para conocer el comportamiento de la red frente a otros optimizadores. Primero, se va a probar con el algoritmo SGD, utilizando los hiperparámetros que se muestran a continuación.

- Taza de aprendizaje = 0.0001
- Número de capas ocultas = 1
- Número de neuronas = 5
- Función de activación = sigmoide
- Optimizador = SGD
- Función de Pérdida = mse
- Número de iteraciones = 30

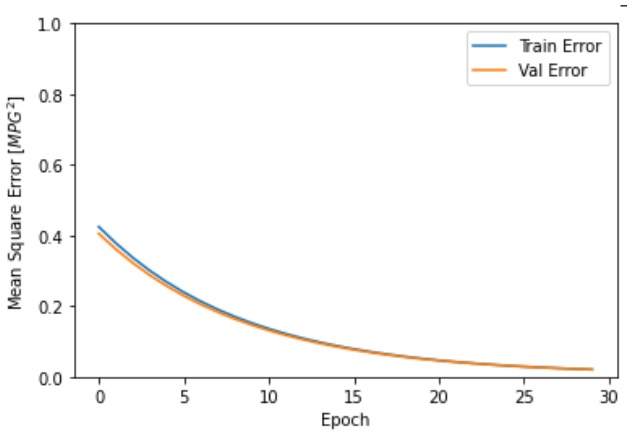

Figura 30. Iteración 7 para la red de estimación de calidad de vino en general.

En este caso, en la figura 30, al igual que en los casos anteriores, se observa que el optimizador “SGD”, lo que hace es ralentizar el proceso de aprendizaje en comparación con el optimizador “Adam”.

Ahora bien, al probar con los optimizadores “Adadelta” y “Adagrad”, al igual que para las redes anteriores, se obtuvo un aprendizaje nulo, ya que el error nunca se ve disminuido.

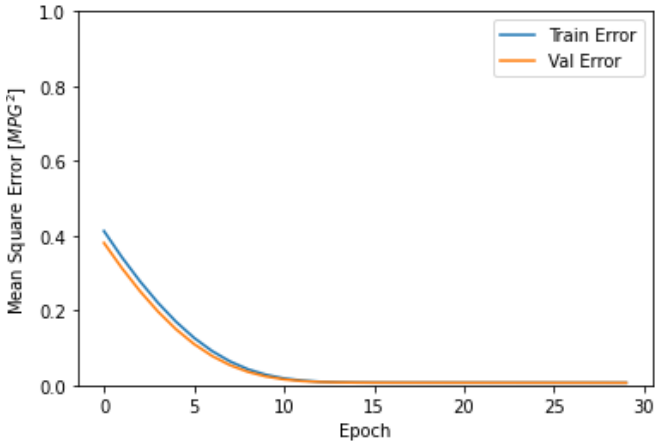

Figura 31. Iteración 8 para la red de estimación de calidad de vino en general.

Finalmente, al probar con el optimizador “Adamax”, se obtiene el comportamiento que se observa en la figura 31, el cual es bastante similar al obtenido con el optimizador “Adam”, sin embargo, se va a utilizar el optimizador “Adam”.

Debido al análisis anterior, se van a utilizar los siguientes hiperparámetros para optimizar el funcionamiento de la red de estimación de calidad de vino en general.

- Taza de aprendizaje = 0.0001
- Número de capas ocultas = 1
- Número de neuronas = 5
- Función de activación = sigmoide
- Optimizador = Adam
- Función de Pérdida = mse
- Número de iteraciones = 15

2- ***<u>Preguntas sobre los modelos neuronales:</u>***

<u>a.</u> Calcule el resultado esperado para las siguientes combinaciones de entrada.

<u>Muestra 1:</u>

<u>CLA</u>:

```
data_prueba = [[7.5, 68, 1, 0, 1, 1, 238], [5.1, 22, 1, 1, 1, 1, 208],[8.2, 81, 0, 0, 1, 1, 257]]
data_pandas = pd.DataFrame(data_prueba, columns = ['pH', 'Temprature', 'Taste', 'Odor', 'Fat ', 'Turbidity', 'Colour'])
data_prueba_norm = normalize(data_pandas)
dat = data_prueba_norm[['pH', 'Temprature', 'Taste', 'Odor', 'Fat ', 'Turbidity', 'Colour']].to_numpy()
print("Datos")
print(dat)

print("Resultados")
results = model.predict(dat).round()

print(results)
```

```
Datos
[[0.69230769 0.67647059 1.         0.         1.         1.
  0.6122449 ]
 [0.32307692 0.         1.         1.         1.         1.
  0.        ]
 [0.8        0.86764706 0.         0.         1.         1.
  1.        ]]
Resultados
[[ 0.  1.  1.]
 [ 0.  0.  0.]
 [ 0.  0.  1.]]
```

Figura 32. Predicción para la clasificación de muestras de leche.

Muestra 2:

VIB:

```
data_prueba = [[5.9, 0.66, 0.58, 2.3, 77, 52, 12, 0.9733, 4.2, 0.56, 10.2],
               [8.7, 0.16, 0.26, 9.3, 25, 16, 34, 0.9933, 3.2, 0.65, 12.2],
               [7.2, 0.64, 0.42, 6.3, 56, 15, 22, 0.9018, 1.2, 1.08, 8.2]]

data_pandas = pd.DataFrame(data_prueba, columns = ['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar', 'chlorides', 'free
total_data = pd.concat([data,data_pandas])
data_prueba_norm = normalize(total_data)
data_norm = data_prueba_norm.tail(3)
dat = data_norm[['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar', 'chlorides', 'free sulfur dioxide', 'total sulfur di
print("Datos")
print(dat)

print("Resultados")
results = model.predict(dat)

print(results*10)
```

```
Datos
[[0.67816092 0.66       0.58       0.24731183 1.         1.
  0.35294118 0.9733     1.         0.51851852 0.83606557]
 [1.         0.16       0.26       1.         0.32467532 0.30769231
  1.         0.9933     0.76190476 0.60185185 1.        ]
 [0.82758621 0.64       0.42       0.67741935 0.72727273 0.28846154
  0.64705882 0.9018     0.28571429 1.         0.67213115]]
Resultados
[[7.8892684]
 [6.2244396]
 [7.202753 ]]
```

Figura 33. Predicción para la calidad de muestra de vino blanco, mediante el modelo de vino blanco.

VINO

```
data_prueba = [[5.9, 0.66, 0.58, 2.3, 77, 52, 12, 0.9733, 4.2, 0.56, 10.2],
               [8.7, 0.16, 0.26, 9.3, 25, 16, 34, 0.9933, 3.2, 0.65, 12.2],
               [7.2, 0.64, 0.42, 6.3, 56, 15, 22, 0.9018, 1.2, 1.08, 8.2]]

data_pandas = pd.DataFrame(data_prueba, columns = ['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar', 'chlorides', 'free
total_data = pd.concat([data,data_pandas])
data_prueba_norm = normalize(total_data)
data_norm = data_prueba_norm.tail(3)
dat = data_norm[['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar', 'chlorides', 'free sulfur dioxide', 'total sulfur di
print("Datos")
print(dat)

print("Resultados")
results = model.predict(dat)

print(results*10)
```

```
Datos
[[0.67816092 0.66       0.58       0.24731183 1.         1.
  0.35294118 0.9733     1.         0.51851852 0.83606557]
 [1.         0.16       0.26       1.         0.32467532 0.30769231
  1.         0.9933     0.76190476 0.60185185 1.        ]
 [0.82758621 0.64       0.42       0.67741935 0.72727273 0.28846154
  0.64705882 0.9018     0.28571429 1.         0.67213115]]
Resultados
[[8.674246 ]
 [7.1501675]
 [7.9310985]]
```

Figura 34. Predicción para la calidad de muestra de vino blanco, mediante el modelo de vino en general.

Muestra 3:

VIT:

```
data_prueba = [[6.3, 0.56, 0.18, 5.3, 177, 17, 23, 0.8933, 8.2, 0.91, 11.7],
               [9.9, 0.26, 0.02, 6.9, 24, 17, 51, 0.8733, 2.3, 0.67, 9.2],
               [5.9, 0.77, 0.12, 7.2, 112, 9, 67, 0.8841, 6.5, 1.17, 14.5]]

data_pandas = pd.DataFrame(data_prueba, columns = ['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar', 'chlorides', 'free
total_data = pd.concat([data,data_pandas])
data_prueba_norm = normalize(total_data)
data_norm = data_prueba_norm.tail(3)
dat = data_norm[['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar', 'chlorides', 'free sulfur dioxide', 'total sulfur di
print("Datos")
print(dat)

print("Resultados")
results = model.predict(dat)

print(results*10)
```

```
Datos
[[0.63636364 0.56       0.18       0.73611111 1.         1.
  0.34328358 0.8933     1.         0.77777778 0.80689655]
 [1.         0.26       0.02       0.95833333 0.13559322 1.
  0.76119403 0.8733     0.2804878  0.57264957 0.63448276]
 [0.5959596  0.77       0.12       1.         0.63276836 0.52941176
  1.         0.8841     0.79268293 1.         1.        ]]
Resultados
[[7.411413 ]
 [7.0013323]
 [6.66929  ]]
```

Figura 35. Predicción para la calidad de muestra de vino tinto, mediante el modelo de vino tinto.

VINO:

```
data_prueba = [[6.3, 0.56, 0.18, 5.3, 177, 17, 23, 0.8933, 8.2, 0.91, 11.7],
               [9.9, 0.26, 0.02, 6.9, 24, 17, 51, 0.8733, 2.3, 0.67, 9.2],
               [5.9, 0.77, 0.12, 7.2, 112, 9, 67, 0.8841, 6.5, 1.17, 14.5]]

data_pandas = pd.DataFrame(data_prueba, columns = ['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar', 'chlorides', 'free
total_data = pd.concat([data,data_pandas])
data_prueba_norm = normalize(total_data)
data_norm = data_prueba_norm.tail(3)
dat = data_norm[['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar', 'chlorides', 'free sulfur dioxide', 'total sulfur di
print("Datos")
print(dat)

print("Resultados")
results = model.predict(dat)

print(results*10)
```

```
Datos
[[0.63636364 0.56       0.18       0.73611111 1.         1.
  0.34328358 0.8933     1.         0.77777778 0.80689655]
 [1.         0.26       0.02       0.95833333 0.13559322 1.
  0.76119403 0.8733     0.2804878  0.57264957 0.63448276]
 [0.5959596  0.77       0.12       1.         0.63276836 0.52941176
  1.         0.8841     0.79268293 1.         1.        ]]
Resultados
[[8.535863 ]
 [8.1702175]
 [7.8617   ]]
```

Figura 36. Predicción para la calidad de muestra de vino tinto, mediante el modelo de vino general.

b. Análisis de los modelos de regresión de vino

Primero, observando los resultados obtenidos en el punto A, referentes al uso de las redes neuronales para obtener la calidad de muestras de vino, se puede observar que,

específicamente para el caso de las muestras de vino blanco, la red diseñada para analizar la calidad de vinos en general tiende a dar una calidad más alta en comparación con la red especifica para los vinos blancos. Sin embargo, las diferencias no son sustancialmente grandes, ya que rondan 1 punto aproximadamente. Sin embargo, para análisis rigurosos de calidad, esto puede ser importante.

Esta diferencia se puede deber a que la red general, se diseñó para considerar aspectos de ambos vinos, por lo cual no puede profundizar en el análisis de muestras específicas, en cambio, la red de vino blanco se diseñó específicamente para muestras de este tipo, por lo que considera cada una de las características (o valores específicos de las características) que hacen que el vino blanco tenga una buena o mala calidad.

Debido a lo anterior, para análisis rigurosos de calidad, es mejor usar la red diseñada específicamente para este tipo de vino, sin embargo, como se mencionó anteriormente, para análisis poco profundos, el modelo general también proporciona resultados aproximados.

Asimismo, para el caso de las muestras de vino tinto, sucede un fenómeno similar, ya que la red para análisis generales tiende a dar una calidad más alta que la red específica para el análisis de vino tinto. Esto, como se mencionó anteriormente se debe a que las características de cada vino son distintas, y debido a que el modelo general debe analizar ambos tipos, no puede realizar esta distinción, ya que, para la red, todas las entradas con las que fue entrenada fueron muestras con el mismo peso, por lo cual no puede profundizar en el análisis de muestras específicas.

Debido a lo anterior, para el caso de los análisis de vino tinto, al igual que para los de vino blanco, se recomienda usar la red específica para este tipo de vino, sin embargo, para análisis poco profundos, la red diseñada para ambos tipos puede dar una aproximación a la calidad real del vino.

Anexos

Anexo 1: Código para la red de clasificación de leche.

```
#Step 1 - Loading the required libraries and modules

from google.colab import drive
drive.mount('/content/drive')

# Import required libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import sklearn

# Import necessary modules
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from math import sqrt

# Keras specific
import tensorflow as tf
import keras
from keras.models import Sequential, Model
from keras.layers import Dense
from keras import Input
from keras.callbacks import LearningRateScheduler

#Step 2 - Reading the Data
data = pd.read_csv('/content/drive/MyDrive/IA/milknew.csv')

#Step 3 - Creating arrays for the features and the response variable

#normalizacion de datos y adaptacion de la columna grade a las salidas de la red
def normalize(data):
  #arrays para las 3 neuronas de salida
  high = []
  medium = []
  low = []
  for column in data:
    if data[column].name == "pH":
      data[column] = (data[column] - 3) / 6.5 #normalización
      continue
    if data[column].name == "Temprature":
      data[column] = (data[column] - 22) / 68 #normalización
      continue
```

```
    if data[column].name == "Taste":
      continue
    if data[column].name == "Odor":
      continue
    if data[column].name == "Fat ":
      continue
    if data[column].name == "Turbidity":
      continue
    if data[column].name == "Colour":
      data[column] = (data[column] - 208) / 49 #normalización
      continue
    if data[column].name == "Grade":
      for element in data[column]:
        if element == 'high':
          high.append(1)
          medium.append(0)
          low.append(0)
        elif element == 'medium':
          high.append(0)
          medium.append(1)
          low.append(0)
        else: #caso low
          high.append(0)
          medium.append(0)
          low.append(1)
    del data['Grade']
    data['High'] = high
    data['Medium'] = medium
    data['Low'] = low
    break
  return data

#datos normalizados
norm_data = normalize(data)
#division de entradas y salidas
features = norm_data[['pH', 'Temprature', 'Taste', 'Odor', 'Fat ', 'Turbidity', 'Colour']].t
o_numpy() #entradas
labels = norm_data[['High', 'Medium', 'Low']].to_numpy() #salidas

#Step 4 - Creating the Training and Test datasets
X_train, X_test, Y_train, Y_test = train_test_split(features, labels, test_size=0.40)

#Step 5 - Define, compile, and fit the Keras classification model

np.random.seed(7)
```

```
tf.random.set_seed(7)

#Definicion del modelo
model = Sequential()
model.add(Input(shape=(7,))) #capa de entrada
model.add(Dense(9, activation='sigmoid'))
model.add(Dense(3)) #capa de salida

#Ajuste de la taza de aprendizaje
lr = LearningRateScheduler(lambda epoch: 0.0001)

# Compilación del modelo
model.compile(optimizer='Adam',
              loss='categorical_crossentropy',
              metrics=['binary_accuracy', 'categorical_crossentropy'])

# Construcción del modelo
mod = model.fit(X_train, Y_train, epochs = 100, validation_data = (X_test, Y_test), callback
s = [lr])

# Resumen de la red
model.summary()

# Obtención del historial de las métricas de la red
hist = pd.DataFrame(mod.history)
hist['epoch'] = mod.epoch

#Construcción de gráficos de error

#Gráfico 1: interaciones vs Perdida (de train y de test)

plt.figure()
plt.xlabel('Epoch')
plt.ylabel('Pérdida')
plt.plot(hist['epoch'], hist['categorical_crossentropy'],
          label='Train Error')
plt.plot(hist['epoch'], hist['val_categorical_crossentropy'],
          label = 'Val Error')
plt.ylim([0,2.5]) #rango en y
plt.legend()

#Mostrar gráficos
plt.show()

#Step 6 - Predictions
```

```
data prueba = [[7.5, 68, 1, 0, 1, 1, 238], [5.1, 22, 1, 1, 1, 1, 208],[8.2, 81, 0, 0, 1, 1,
257]]
data_pandas = pd.DataFrame(data_prueba, columns = ['pH', 'Temprature', 'Taste', 'Odor', 'Fat
 ', 'Turbidity', 'Colour'])
data prueba norm = normalize(data pandas)
dat = data_prueba_norm[['pH', 'Temprature', 'Taste', 'Odor', 'Fat ', 'Turbidity', 'Colour']]
.to_numpy()
print("Datos")
print(dat)

print("Resultados")
results = model.predict(dat).round()

print(results)
```

Anexo 2: Código para la red de estimación de calidad de vino blanco.

```
#Step 1 - Loading the required libraries and modules
from google.colab import drive
drive.mount('/content/drive')

# Import required libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import sklearn

# Import necessary modules
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from math import sqrt

# Keras specific
import tensorflow as tf
import keras
from keras.models import Sequential, Model
from keras.layers import Dense
from keras import Input
from keras.callbacks import LearningRateScheduler

#Step 2 - Reading the Data
data = pd.read_csv('/content/drive/MyDrive/IA/winequality-white.csv', sep = ';')
#Step 3 - Creating arrays for the features and the response variable
def normalize(data):
  for column in data:
```

```
    if data[column].name == 'quality':
      data[column] = data[column]/10
      break
    else:
      data[column] = (data[column] - data[column].min()) / (data[column].max() - data[column
].min())
  return data

data_norm = normalize(data)

labels = data_norm[['quality']].to_numpy() #salidas
features = data_norm[['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar',
'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'density', 'pH', 'sulphates', 'a
lcohol']].to_numpy() #entradas

#Step 4 - Creating the Training and Test datasets
X_train, X_test, Y_train, Y_test = train_test_split(features, labels, test_size=0.40)

#Step 5 - Define, compile, and fit the Keras classification model
np.random.seed(7)
tf.random.set_seed(7)

model = Sequential()
model.add(Input(shape=(11,))) #capa de entrada
model.add(Dense(5, activation='sigmoid'))
model.add(Dense(1)) #capa de salida

#Adjusting the learning rate

lr = LearningRateScheduler(lambda epoch: 0.0001)

# Compile the model
model.compile(optimizer='adam',
              loss='mse',
              metrics=['mae','mse'])

# build the model
mod = model.fit(X_train, Y_train, epochs = 20, validation_data = (X_test, Y_test), callbacks
 = [lr])
model.summary()

print(mod.history)

hist = pd.DataFrame(mod.history)
hist['epoch'] = mod.epoch
```

```
plt.figure()
plt.xlabel('Epoch')
plt.ylabel('Mean Abs Error [MPG]')
plt.plot(hist['epoch'], hist['mae'],
         label='Train Error')
plt.plot(hist['epoch'], hist['val_mae'],
         label = 'Val Error')
plt.ylim([0,2])
plt.legend()

plt.figure()
plt.xlabel('Epoch')
plt.ylabel('Mean Square Error [$MPG^2$]')
plt.plot(hist['epoch'], hist['mse'],
         label='Train Error')
plt.plot(hist['epoch'], hist['val_mse'],
         label = 'Val Error')
plt.ylim([0,1])
plt.legend()
plt.show()

#Step 6 - Predictions
data prueba = [[5.9, 0.66, 0.58, 2.3, 77, 52, 12, 0.9733, 4.2, 0.56, 10.2],
               [8.7, 0.16, 0.26, 9.3, 25, 16, 34, 0.9933, 3.2, 0.65, 12.2],
               [7.2, 0.64, 0.42, 6.3, 56, 15, 22, 0.9018, 1.2, 1.08, 8.2]]

data_pandas = pd.DataFrame(data_prueba, columns = ['fixed acidity', 'volatile acidity', 'cit
ric acid', 'residual sugar', 'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'de
nsity', 'pH', 'sulphates', 'alcohol'])
total_data = pd.concat([data,data_pandas])
data_prueba_norm = normalize(total_data)
data_norm = data_prueba_norm.tail(3)
dat = data_norm[['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar', 'chlo
rides', 'free sulfur dioxide', 'total sulfur dioxide', 'density', 'pH', 'sulphates', 'alcoho
l']].to_numpy()
print("Datos")
print(dat)

print("Resultados")
results = model.predict(dat)

print(results*10)
```

Anexo 3: Código para la red de estimación de calidad de vino tinto.

```
#Step 1 - Loading the required libraries and modules
from google.colab import drive
drive.mount('/content/drive')

# Import required libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import sklearn

# Import necessary modules
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from math import sqrt

# Keras specific
import tensorflow as tf
import keras
from keras.models import Sequential, Model
from keras.layers import Dense
from keras import Input
from keras.callbacks import LearningRateScheduler

#Step 2 - Reading the Data
data = pd.read_csv('/content/drive/MyDrive/IA/winequality-red.csv', sep = ';')

#Step 3 - Creating arrays for the features and the response variable
def normalize(data):
  for column in data:
    if data[column].name == 'quality':
      data[column] = data[column]/10
      break
    else:
      data[column] = (data[column] - data[column].min()) / (data[column].max() - data[column
].min())
  return data

data_norm = normalize(data)

labels = data_norm[['quality']].to_numpy() #salidas
features = data_norm[['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar',
'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'density', 'pH', 'sulphates', 'a
lcohol']].to_numpy() #entradas

#Step 4 - Creating the Training and Test datasets
```

```
X train, X test, Y train, Y test = train test split(features, labels, test size=0.40)

#Step 5 - Define, compile, and fit the Keras classification model
np.random.seed(7)
tf.random.set seed(7)

model = Sequential()
model.add(Input(shape=(11,))) #capa de entrada
model.add(Dense(5, activation='sigmoid'))
model.add(Dense(1)) #capa de salida

#Adjusting the learning rate

lr = LearningRateScheduler(lambda epoch: 1e-3)

# Compile the model
model.compile(optimizer='Adam',
              loss='mse',
              metrics=['mae','mse'])

# build the model
mod = model.fit(X_train, Y_train, epochs = 10, validation_data = (X_test, Y_test), callbacks
 = [lr])
model.summary()

print(mod.history)

hist = pd.DataFrame(mod.history)
hist['epoch'] = mod.epoch

plt.figure()
plt.xlabel('Epoch')
plt.ylabel('Mean Abs Error [MPG]')
plt.plot(hist['epoch'], hist['mae'],
          label='Train Error')
plt.plot(hist['epoch'], hist['val_mae'],
          label = 'Val Error')
plt.ylim([0,1])
plt.legend()

plt.figure()
plt.xlabel('Epoch')
plt.ylabel('Mean Square Error [$MPG^2$]')
plt.plot(hist['epoch'], hist['mse'],
          label='Train Error')
```

```
plt.plot(hist['epoch'], hist['val mse'],
          label = 'Val Error')
plt.ylim([0,1])
plt.legend()
plt.show()

#Step 6 - Predictions
data_prueba = [[6.3, 0.56, 0.18, 5.3, 177, 17, 23, 0.8933, 8.2, 0.91, 11.7],
               [9.9, 0.26, 0.02, 6.9, 24, 17, 51, 0.8733, 2.3, 0.67, 9.2],
               [5.9, 0.77, 0.12, 7.2, 112, 9, 67, 0.8841, 6.5, 1.17, 14.5]]

data_pandas = pd.DataFrame(data_prueba, columns = ['fixed acidity', 'volatile acidity', 'cit
ric acid', 'residual sugar', 'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'de
nsity', 'pH', 'sulphates', 'alcohol'])
total_data = pd.concat([data,data_pandas])
data prueba norm = normalize(total data)
data_norm = data_prueba_norm.tail(3)
dat = data_norm[['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar', 'chlo
rides', 'free sulfur dioxide', 'total sulfur dioxide', 'density', 'pH', 'sulphates', 'alcoho
l']].to numpy()
print("Datos")
print(dat)

print("Resultados")
results = model.predict(dat)

print(results*10)
```

Anexo 4: Código para la red de estimación general de calidad de vino.

```
#Step 1 - Loading the required libraries and modules
from google.colab import drive
drive.mount('/content/drive')

# Import required libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import sklearn

# Import necessary modules
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from math import sqrt
```

```
# Keras specific
import tensorflow as tf
import keras
from keras.models import Sequential, Model
from keras.layers import Dense
from keras import Input
from keras.callbacks import LearningRateScheduler

#Step 2 - Reading the Data
data = pd.read_csv('/content/drive/MyDrive/IA/winequality-general.csv', sep = ';')

#Step 3 - Creating arrays for the features and the response variable
def normalize(data):
  for column in data:
    if data[column].name == 'quality':
      data[column] = data[column]/10
      break
    else:
      data[column] = (data[column] - data[column].min()) / (data[column].max() - data[column
].min())
  return data

data_norm = normalize(data)
print(data norm)

labels = data_norm[['quality']].to_numpy() #salidas
features = data_norm[['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar',
'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'density', 'pH', 'sulphates', 'a
lcohol']].to_numpy() #entradas

#Step 4 - Creating the Training and Test datasets
X_train, X_test, Y_train, Y_test = train_test_split(features, labels, test_size=0.40)

#Step 5 - Define, compile, and fit the Keras classification model
np.random.seed(7)
tf.random.set_seed(7)

model = Sequential()
model.add(Input(shape=(11,))) #capa de entrada
model.add(Dense(5, activation='sigmoid'))
model.add(Dense(1)) #capa de salida

#Adjusting the learning rate

lr = LearningRateScheduler(lambda epoch: 1e-4)
```

```
# Compile the model
model.compile(optimizer='Adam',
              loss='mse',
              metrics=['mae','mse'])

# build the model
mod = model.fit(X_train, Y_train, epochs = 30, validation_data = (X_test, Y_test), callbacks
 = [lr])
model.summary()

print(mod.history)

hist = pd.DataFrame(mod.history)
hist['epoch'] = mod.epoch

plt.figure()
plt.xlabel('Epoch')
plt.ylabel('Mean Abs Error [MPG]')
plt.plot(hist['epoch'], hist['mae'],
          label='Train Error')
plt.plot(hist['epoch'], hist['val_mae'],
          label = 'Val Error')
plt.ylim([0,1])
plt.legend()

plt.figure()
plt.xlabel('Epoch')
plt.ylabel('Mean Square Error [$MPG^2$]')
plt.plot(hist['epoch'], hist['mse'],
          label='Train Error')
plt.plot(hist['epoch'], hist['val_mse'],
          label = 'Val Error')
plt.ylim([0,1])
plt.legend()
plt.show()

#Step 6: Predictions

#Vino Blanco

data_prueba = [[5.9, 0.66, 0.58, 2.3, 77, 52, 12, 0.9733, 4.2, 0.56, 10.2],
               [8.7, 0.16, 0.26, 9.3, 25, 16, 34, 0.9933, 3.2, 0.65, 12.2],
               [7.2, 0.64, 0.42, 6.3, 56, 15, 22, 0.9018, 1.2, 1.08, 8.2]]
```

```
data pandas = pd.DataFrame(data prueba, columns = ['fixed acidity', 'volatile acidity', 'cit
ric acid', 'residual sugar', 'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'de
nsity', 'pH', 'sulphates', 'alcohol'])
total_data = pd.concat([data,data_pandas])
data prueba norm = normalize(total data)
data_norm = data_prueba_norm.tail(3)
dat = data_norm[['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar', 'chlo
rides', 'free sulfur dioxide', 'total sulfur dioxide', 'density', 'pH', 'sulphates', 'alcoho
l']].to numpy()
print("Datos")
print(dat)

print("Resultados")
results = model.predict(dat)

print(results*10)

#Vino tinto

data prueba = [[6.3, 0.56, 0.18, 5.3, 177, 17, 23, 0.8933, 8.2, 0.91, 11.7],
               [9.9, 0.26, 0.02, 6.9, 24, 17, 51, 0.8733, 2.3, 0.67, 9.2],
               [5.9, 0.77, 0.12, 7.2, 112, 9, 67, 0.8841, 6.5, 1.17, 14.5]]

data pandas = pd.DataFrame(data prueba, columns = ['fixed acidity', 'volatile acidity', 'cit
ric acid', 'residual sugar', 'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'de
nsity', 'pH', 'sulphates', 'alcohol'])
total_data = pd.concat([data,data_pandas])
data_prueba_norm = normalize(total_data)
data_norm = data_prueba_norm.tail(3)
dat = data_norm[['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar', 'chlo
rides', 'free sulfur dioxide', 'total sulfur dioxide', 'density', 'pH', 'sulphates', 'alcoho
l']].to_numpy()
print("Datos")
print(dat)

print("Resultados")
results = model.predict(dat)

print(results*10)
```