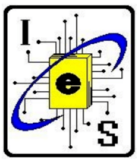

ACTIVIDAD DE TRABAJO COLABORATIVO

ANÁLISIS DEL PERCEPTRÓN MULTICAPA

PRESENTADO POR:
CRISTIAN JAVIER ARENAS COGOLLO

DIRIGIDO A: DOCENTE MANUEL ALEJANDRO OSPINA ALARCÓN

INTELIGENCIA ARTIFICIAL

UNIVERSIDAD DE CARTAGENA

FACULTAD DE INGENIERÍA

PROGRAMA DE INGENIERÍA DE SISTEMAS

CARTAGENA DE INDIAS D.T Y C - BOLÍVAR COLOMBIA

2024

Tabla de contenido

Descripción del problema

Tenemos una población que dentro de un tiempo determinado se registró su consumo eléctrico (en MegaWatts) durante las 24 horas del día en los 7 días de la semana (24/7). Estos datos representan el consumo de energía de la población, la cual en algunos momentos del día alcanza picos muy altos lo cual podría ser un riesgo al congestionar el flujo eléctrico ver figura 1. Por lo tanto, con los datos obtenidos se desea tomar medidas para que en un futuro tener presente cuales son los momentos del día con mayor tráfico y así la empresa encargada de suministrar la electricidad a la población y las entes gubernamentales, puedan prevenir posibles accidentes.

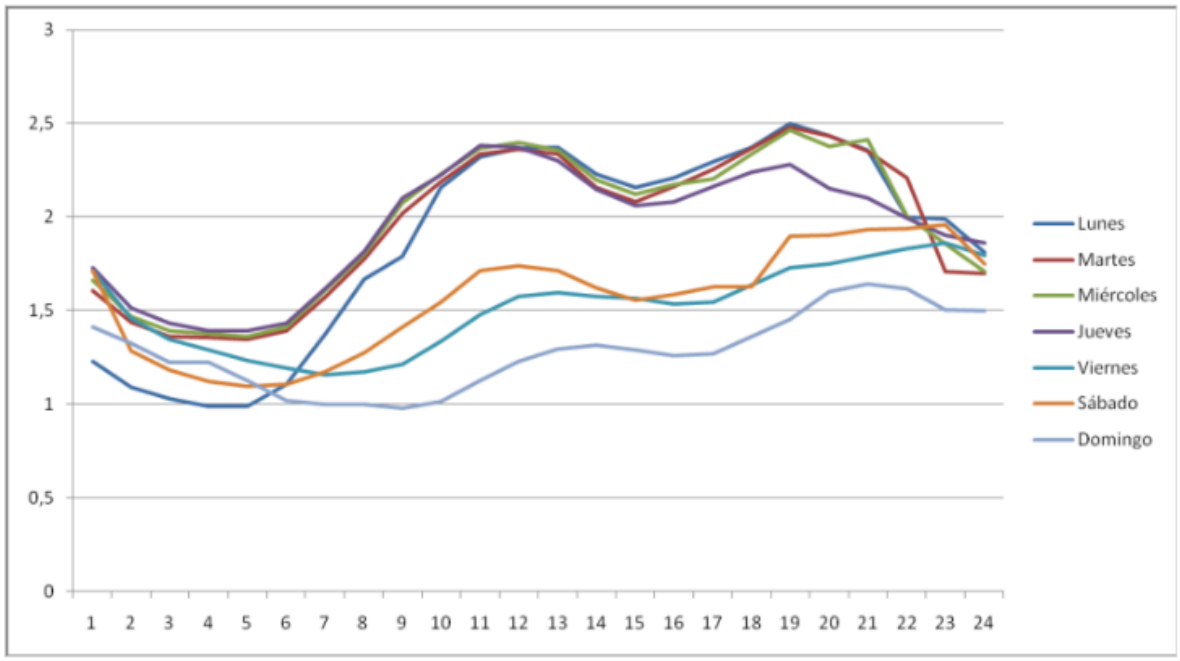

Figura 1: Comportamiento del flujo eléctrico en una población

Fuente: Material de trabajo proporcionado por el docente del curso

Propuesta de la solución

Para aportar una solución al problema se propone utilizar una red neuronal multicapa con la cual a partir de los datos del consumo de la población, esta pueda predecir cómo será el comportamiento eléctrico en una hora y día determinado; esto será útil para empresas eléctricas y el gobierno, pues con la información entregada por la red MLP, es posible aplicar regulaciones en la población con el fin de dar un buen uso al flujo eléctrico, tanto para reducir costos a las personas en su factura, evitar el riesgo de que los transformadores se exploten, y sobretodo para reducir el impacto ambiental que genera sobrecargar una red eléctrica.

Mediante el lenguaje Python, aprovechando sus librerías Numpy, y Matplotlib, se realizará la red neuronal multicapa, la cual a partir de un archivo .txt, que contenga los datos del consumo eléctrico de una población, donde las filas corresponden a las horas del día y las columnas a los días de la semana, para luego dar la libertad al usuario desarrollador, de configurar la arquitectura de la red, definiendo sus entradas, salidas y sus capas internas, con su número de neuronas, esto con el fin de que dependiendo del contexto del problema y las preferencias del usuario, se pueda adaptar a las necesidades de este y así alcanzar una precisión y exactitud en los resultados.

Definición de Datos esperados

Para este problema se tiene como entrada el número de horas del día (1 a 24), y uno de los 7 días de la semana (1 al 7), a partir de esto se deberá entregar como salida un valor aproximado del consumo esperado bajo los parámetros suministrados.

1. ¿Cuál es la salida esperada?: 1 salida, que corresponde al consumo esperado en un día y hora determinada.
2. ¿Cuáles serían los datos de entrada?: 2 entradas, el día de la semana y una hora de ese día.

Arquitecturas de la red neuronal

Debido a que el problema cuenta con 2 entradas y una salida, se mantendrá esta correspondencia para las capas de entrada y salida, dejando a disposición del usuario cuantas neuronas y capas ocultas usar. Para esta situación, se proponen 3 configuraciones.

Propuestas de arquitectura

A. 2:7:24:1 → Para un patrón de entrada de dos dimensiones, se tiene una sola salida de red; se tienen 7 neuronas en la primera capa oculta y 24 neuronas en la segunda capa oculta

B. 2:12:8:1 → Para un patrón de entrada de dos dimensiones, se tiene una sola salida de red, se tienen 12 neuronas en la primera capa oculta y 8 neuronas en la segunda capa oculta

C. 2:10:10:1 → Para un patrón de entrada de dos dimensiones, se tiene una sola salida de red, se tienen 10 neuronas en la primera capa oculta y 10 neuronas en la segunda capa oculta

Implementación del algoritmo

El código está repartido en tres secciones, una para leer los datos del consumo eléctrico de un archivo .txt, que luego será normalizado en valores entre 0 y 1; posteriormente se procede a definir el cuerpo de la red, mediante la implementación de la función de activación y su derivada, luego definir el algoritmo de propagación hacia adelante y hacia atrás, para luego establecer la función para el entrenamiento del perceptrón multicapa. Por último, cuenta con la salida de datos, donde se proporciona una entrada por consola para el día y la hora, y así obtener la predicción de ese momento, luego se acompaña de una gráfica para mostrar el comportamiento del error de la red, hasta unas 1000 iteraciones propuestas, y por último, una gráfica que presenta el valor del consumo real con el estimado por la red, lo cual ayudará a determinar qué tan preciso y exacto es la respuesta del MLP.

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

# lectura del archivo

# funcion para leer un archivo txt
def leer_datos(archivo):
    with open(archivo, 'r') as f:
        lineas = f.readlines()

    datos = [list(map(float, linea.strip().split())) for linea in lineas]

    return np.array(datos)

#normalizar los valores a la escala de 0 y 1 dividiendo entre 3.0 los
valores del .txt
def normalizar_datos(datos):
    return datos / 3.0
```

```
#ejecucion
# se define el archivo y se llama a la función que lo abre
archivo = "datosMLP.txt"
datos = leer_datos(archivo)

# se llama a la función para normalizar los datos
datos_normalizados = normalizar_datos(datos)

# preparar datos y arquitectura
dias_semana = np.arange(0,7)
horas_dia = np.arange(1,25)

#combinaciones
entradas = []
salidas = []

for hora in range(24):
    for dia in range(7):
        entradas.append([hora, dia])
        salidas.append(datos_normalizados[hora, dia])

entradas = np.array(entradas)
salidas = np.array(salidas).reshape(-1, 1)
```

Figura 2: Bloque de código para el proceso de lectura y normalización de un archivo de datos txt

Fuente: Elaboración propia

Python

```
# Definicion de la clase de la red MLP
class PerceptronMLP:
    def __init__(self, estructura):
        self.pesos = []
        self.sesgos = []
        self.activaciones = []

        for i in range(len(estructura)-1):
```

```
            peso = np.random.rand(estructura[i], estructura[i+1]) - 0.5
            sesgo = np.random.rand(1,estructura[i+1]) - 0.5
            self.pesos.append(peso)
            self.sesgos.append(sesgo)
#funcion de activacion sigmoide
    def sigmoid(self, x):
        return 1/(1+np.exp(-x))

#derivada de la funcion sigmoide
    def sigmoid_derivada(self, x):
        return x * (1-x)

#algoritmo de propagación hacia adelante
    def propagacion(self, entrada):
        self.activaciones = [entrada]
        for peso, sesgo in zip(self.pesos, self.sesgos):
            z = np.dot(self.activaciones[-1], peso) + sesgo
            a = self.sigmoid(z)
            self.activaciones.append(a)
        return self.activaciones[-1]

#algoritmo de propagación hacia atrás (retropropagación)
    def retropropagacion(self, salida_esperada, tasa_aprendizaje):
        errores = [salida_esperada - self.activaciones[-1]]
        for i in reversed(range(len(self.pesos))):
            error = errores[-1]
            delta = error * self.sigmoid_derivada(self.activaciones[i+1])
            errores.append(np.dot(delta, self.pesos[i].T))
            self.pesos[i] += np.dot(self.activaciones[i].T, delta) *
tasa_aprendizaje
            self.sesgos[i] += np.sum(delta, axis = 0, keepdims=True) *
tasa_aprendizaje
        errores.reverse()
        return errores[0]

#función para el entrenamiento de la red
    def entrenar(self, datos_entrada, datos_salida, iteraciones,
tasa_aprendizaje):
        errores_totales = []
        for _ in range(iteraciones):
            salida_predicha = self.propagacion(datos_entrada)
```

```
            error_actual = self.retropropagacion(datos_salida,
tasa_aprendizaje)
            error_promedio = np.mean(np.abs(error_actual))
            errores_totales.append(error_promedio)
        return errores_totales

#funcion para recibir entradas y regresar la salida esperada (la predicción
de la red)
    def predecir(self, dia, hora):
        entrada = np.array([[hora, dia]])
        prediccion = self.propagacion(entrada)
        return prediccion[0][0]
```

Figura 3: Bloque de código para la definición de la clase PerceptronMLP con sus respectivos métodos

Fuente: Elaboración propia

Python

```
#inicio de las pruebas
# definir la estructura
estructura = [2,7,24,1] #las capas se adaptan en base a lo que
desarrollador necesite y crea pertienente
perceptron = PerceptronMLP(estructura)
iteraciones = 10000
tasa_aprendizaje = 0.01

#ingreso de variables de entrada para proponer y lograr una predicción
dia_usuario = int(input("Ingrese un dia de la semana (0 a 6): "))
hora_usuario = int(input("Ingrese la hora del día (1 a 24): "))

#entrenar la red
errores_totales = perceptron.entrenar(entradas, salidas, iteraciones,
tasa_aprendizaje)

#Grafica 1: error total de la red durante el aprendizaje
plt.figure(figsize=(10,6))
plt.plot(errores_totales)
plt.title('Evolución del error durante el entrenamiento')
```

```
plt.xlabel('Iteraciones')
plt.ylabel('Error promedio absoluto')
plt.grid(True)
plt.show()

#obtener una prediccion para la grafica 2
predicciones = perceptron.propagacion(entradas)

#grafica 2: consumo real vs consumo estimado
plt.figure(figsize=(10, 6))

# Consumo real (valores del archivo .txt)
plt.plot(salidas, label='Consumo Real', color='blue')

# Consumo predicho por la red neuronal
plt.plot(predicciones, label='Consumo Predicho', color='red',
linestyle='--')

# Detalles de la gráfica
plt.title('Consumo Real vs Consumo estimado')
plt.xlabel('Muestras (Combinaciones de Día y Hora)')
plt.ylabel('Consumo Eléctrico')
plt.legend()
plt.grid(True)
plt.show()

prediccion_usuario = perceptron.predecir(dia_usuario, hora_usuario)
print(f"El consumo electrico en el dia {dia_usuario} a las
{hora_usuario}:00 es: {prediccion_usuario:.4f}")
```

Figura 4: Bloque de código para la inicialización de la arquitectura y la impresión de las gráficas

Fuente: Elaboración propia

Gráficos y resultados

Con el fin de mostrar el comportamiento y entrenamiento de la red neuronal multicapa, en cada una de las distintas arquitecturas, se va a realizar 2 pruebas por cada configuración, por lo tanto se harán 6 pruebas que cada par llevaría los mismo valores de entrada para ver cómo varían los resultados y su comportamiento.

Cabe mencionar que, para la gráfica de consumo real y estimado, se realiza una combinación de los días y horas, por lo que en el eje X, hay 7 separaciones con unidades de 25 como diferencia entre separaciones.

Probando arquitectura A

2:7:24:1

Se utiliza este patrón siguiendo la lógica de la matriz que se forma con los datos del comportamiento eléctrico, 7 días de la semana, con 24 horas del día.

- Se dejará un límite de 10000 iteraciones
- La tasa de aprendizaje será de 0,01
- Para las entradas del usuario, se tomará el día 3 (jueves) y la hora 18 (6:00 pm)

Prueba 1

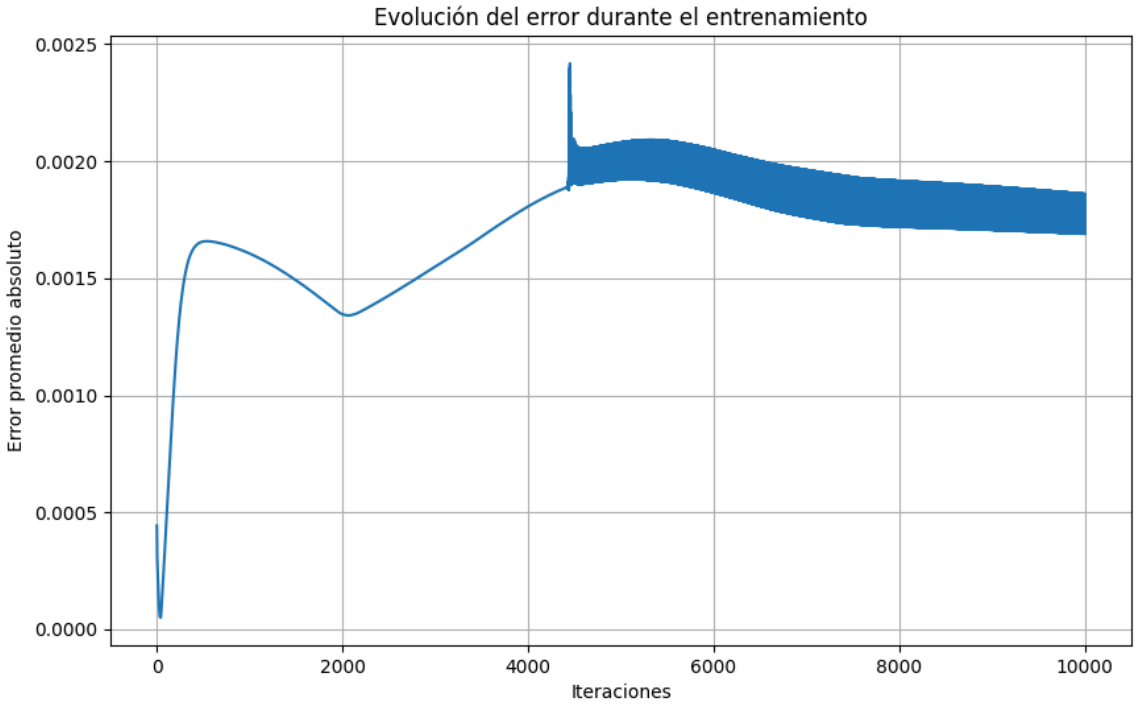

Figura 5: Comportamiento del error de la red durante el entrenamiento de la prueba 1

Fuente: Elaboración propia

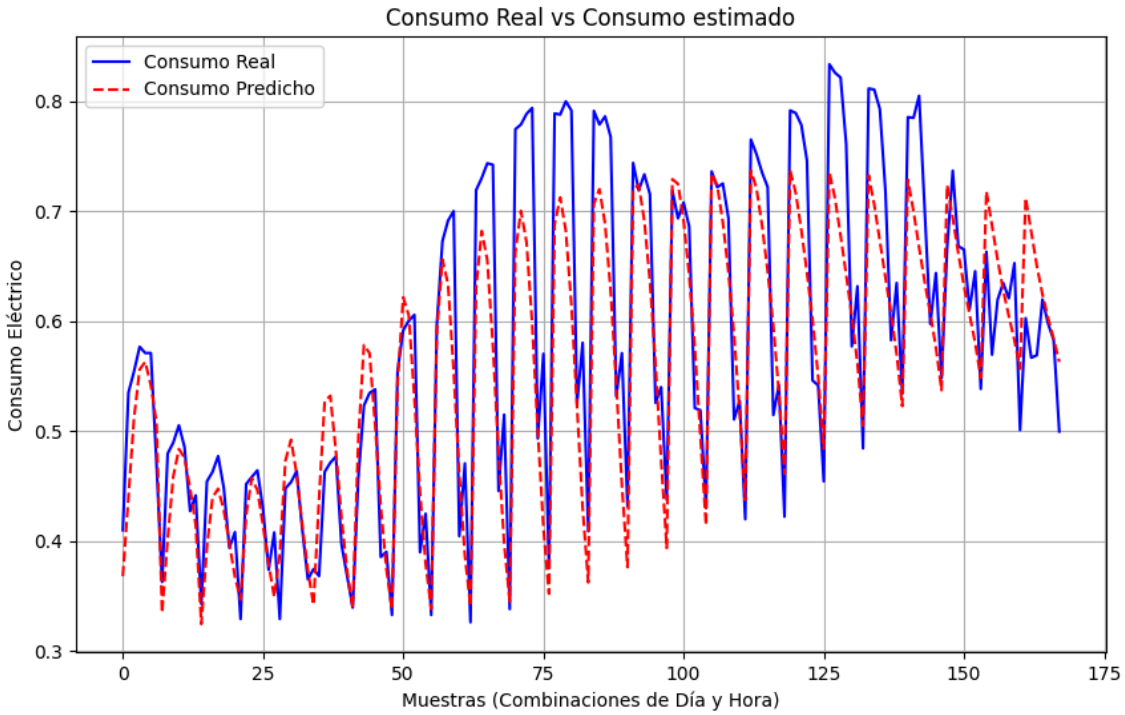

Figura 6: Consumo real vs estimado por la red en la prueba 1

Fuente: Elaboración propia

```
Ingrese un dia de la semana (0 a 6): 3
Ingrese la hora del día (1 a 24): 18
El consumo electrico en el dia 3 a las 18:00 es: 0.6420
```

Figura 7: Resultados por consola de la prueba 1

Fuente: Elaboración propia

Prueba 2

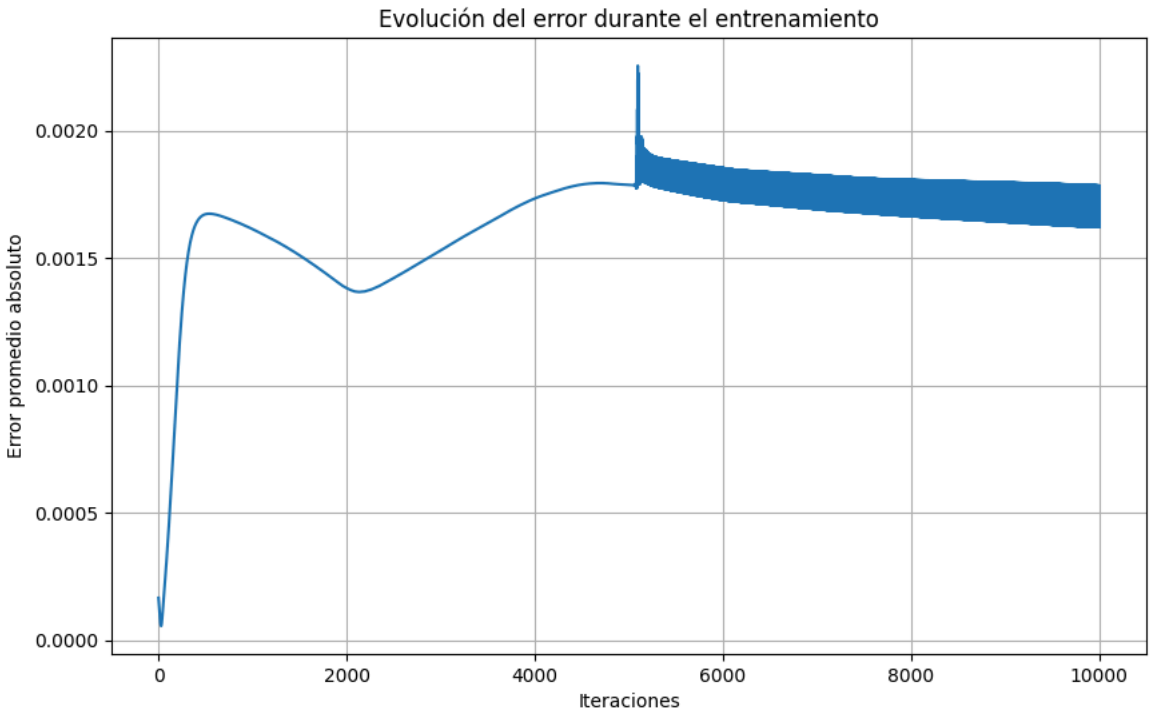

Figura 8: Comportamiento del error de la red durante el entrenamiento de la prueba 2

Fuente: Elaboración propia

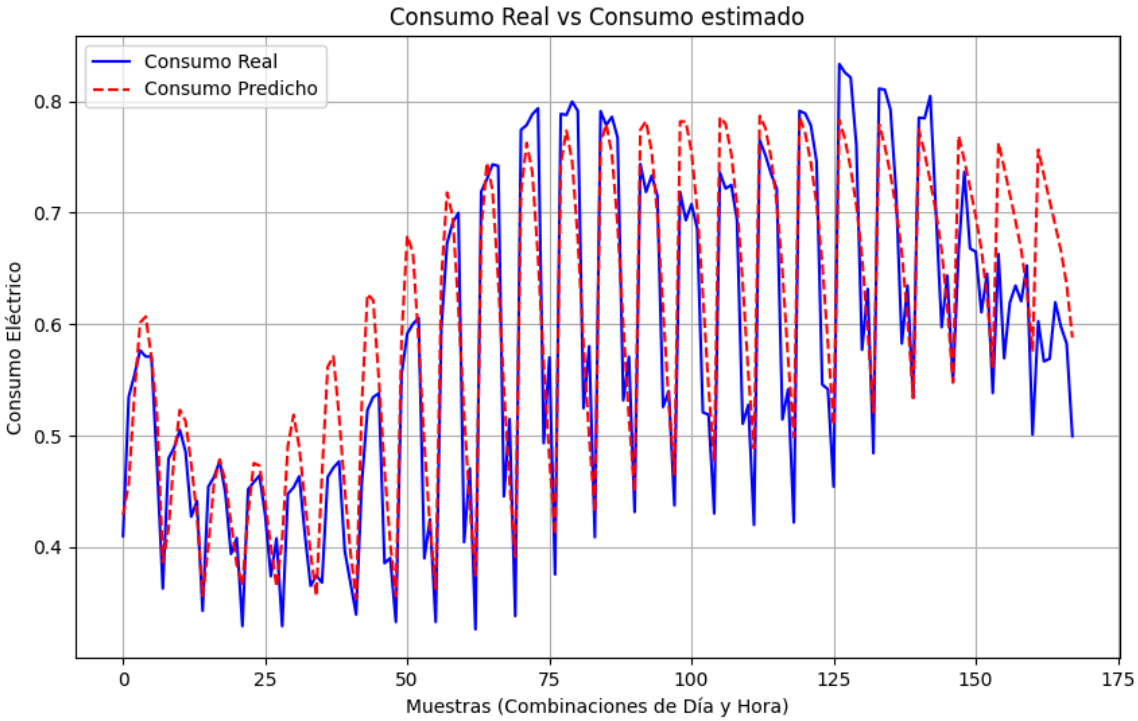

Figura 9: Consumo real vs estimado por la red en la prueba 2

Fuente: Elaboración propia

```
Ingrese un dia de la semana (0 a 6): 3
Ingrese la hora del día (1 a 24): 18
El consumo electrico en el dia 3 a las 18:00 es: 0.7066
```

Figura 10: Resultados por consola de la prueba 2

Fuente: Elaboración propia

Probando arquitectura B

2:12:8:1

Se utiliza este patrón siguiendo las indicaciones del material de trabajo proporcionado por el docente

- Se dejará un límite de 10000 iteraciones
- La tasa de aprendizaje será de 0,01
- Para las entradas del usuario, se tomará el día 6 (domingo) y la hora 21 (9:00 pm)

Prueba 1

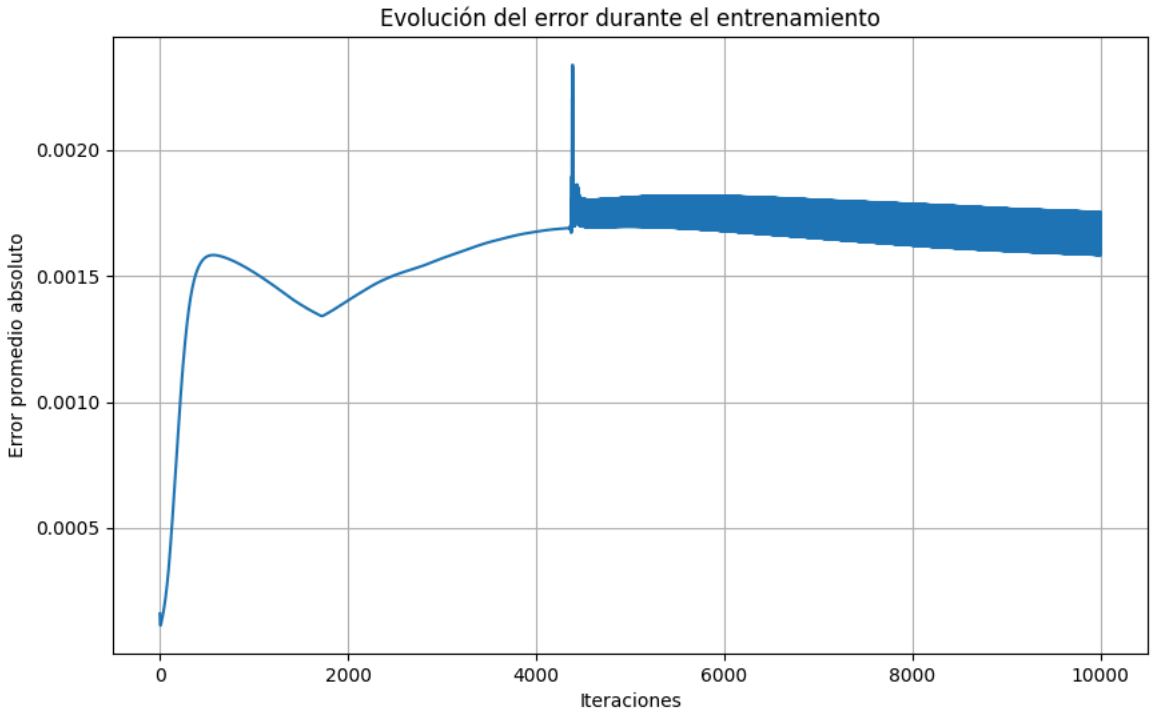

Figura 11: Comportamiento del error de la red durante el entrenamiento de la prueba 1

Fuente: Elaboración propia

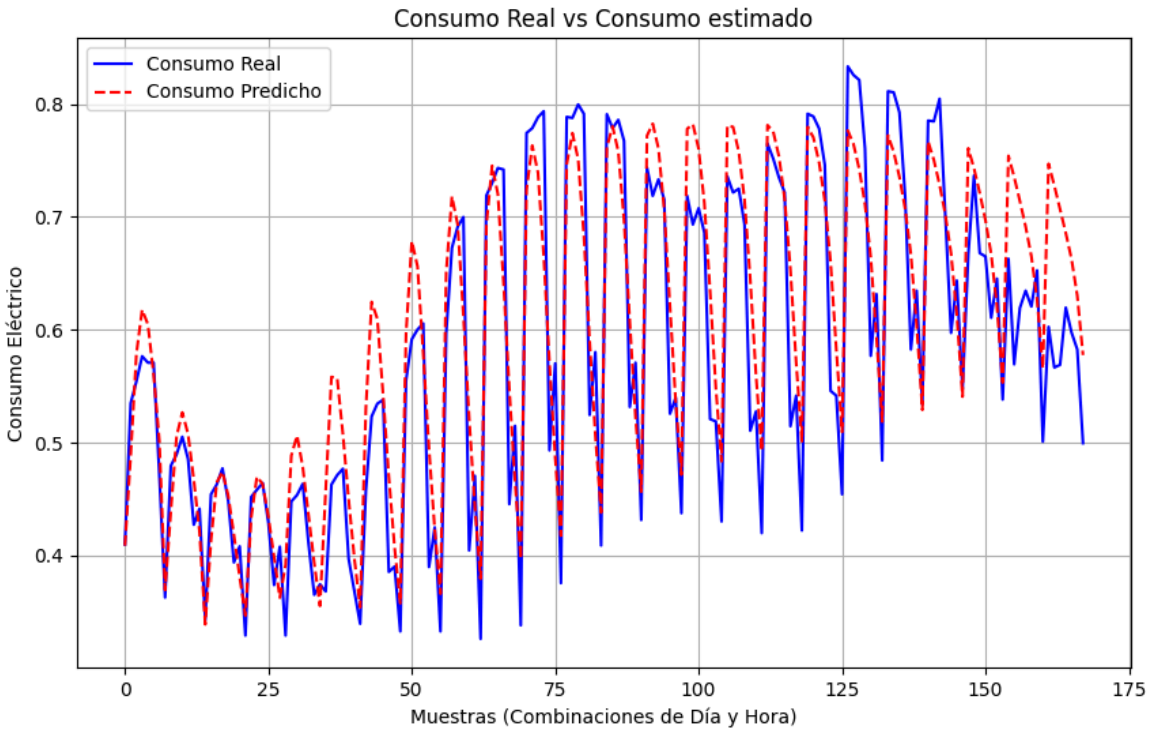

Figura 12: Consumo real vs estimado por la red en la prueba 1

Fuente: Elaboración propia

```
Ingrese un dia de la semana (0 a 6): 6
Ingrese la hora del día (1 a 24): 21
El consumo electrico en el dia 6 a las 21:00 es: 0.5533
```

Figura 13: Resultados por consola de la prueba 1

Fuente: Elaboración propia

Prueba 2

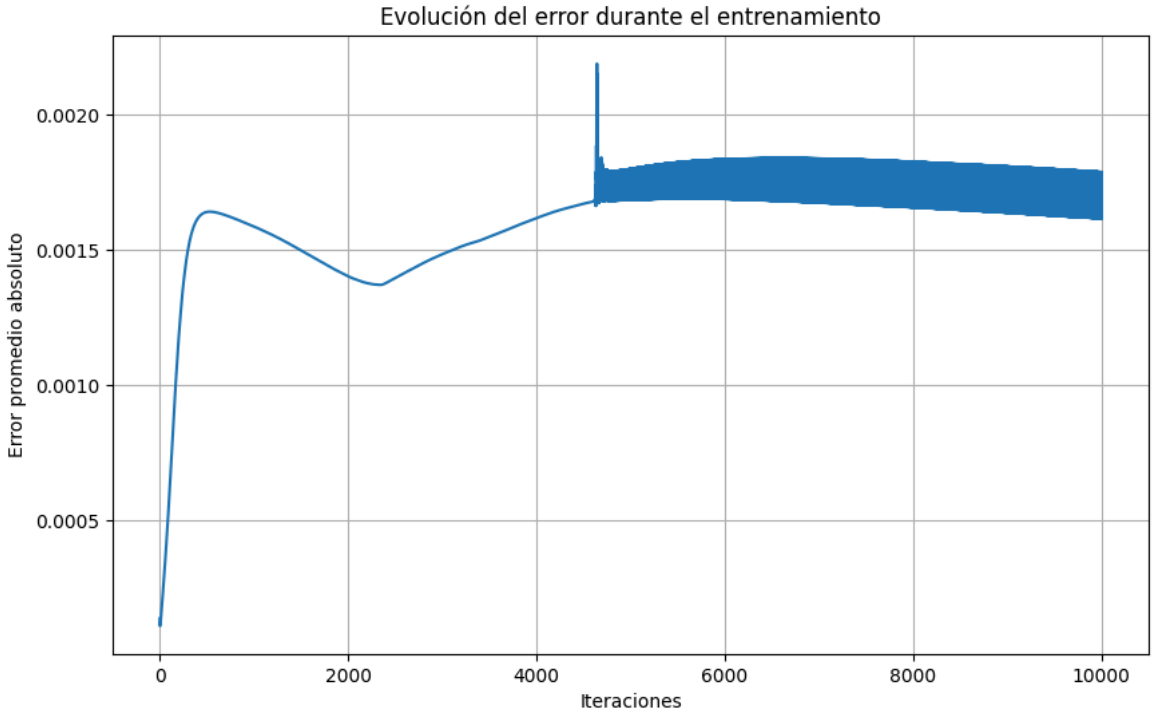

Figura 14: Comportamiento del error de la red durante el entrenamiento de la prueba 2

Fuente: Elaboración propia

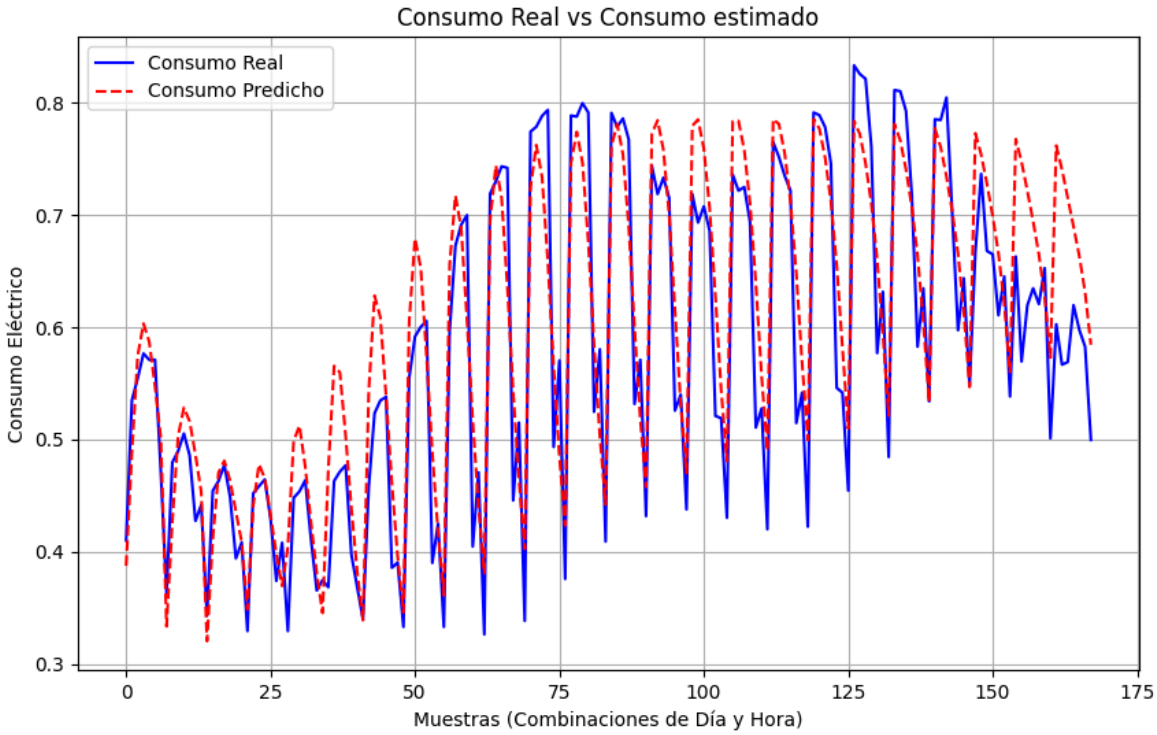

Figura 15: Consumo real vs estimado por la red en la prueba 2

Fuente: Elaboración propia

```
Ingrese un dia de la semana (0 a 6): 6
Ingrese la hora del día (1 a 24): 21
El consumo electrico en el dia 6 a las 21:00 es: 0.5602
```

Figura 16: Resultados por consola de la prueba 2

Fuente: Elaboración propia

Probando arquitectura C

2:10:10:1

Se utiliza este patrón buscando mantener una simetría entre las capas, aprovechando que cuentan con el mismo número de neuronas.

- Se dejará un límite de 10000 iteraciones
- La tasa de aprendizaje será de 0,01
- Para las entradas del usuario, se tomará el día 1 (lunes) y la hora 7 (7:00 am)

Prueba 1

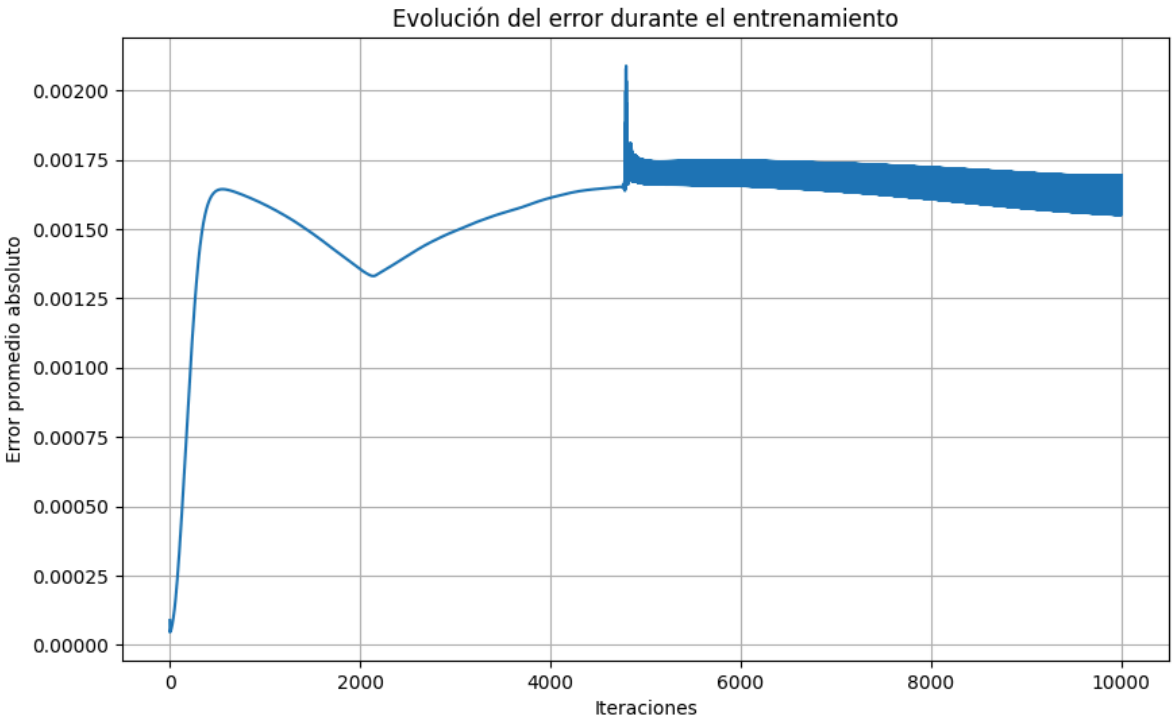

Figura 17: Comportamiento del error de la red durante el entrenamiento de la prueba 1

Fuente: Elaboración propia

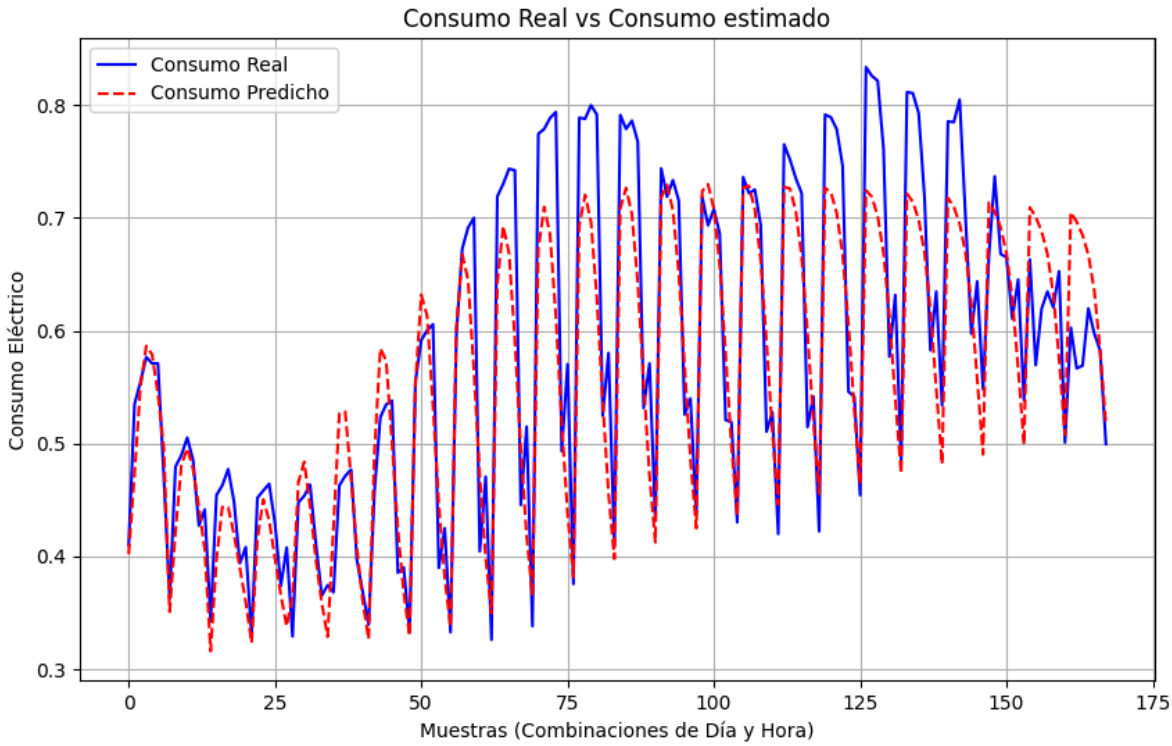

Figura 18: Consumo real vs estimado por la red en la prueba 1

Fuente: Elaboración propia

```
Ingrese un dia de la semana (0 a 6): 1
Ingrese la hora del día (1 a 24): 7
El consumo electrico en el dia 1 a las 7:00 es: 0.6318
```

Figura 19: Resultados por consola de la prueba 1

Fuente: Elaboración propia

Prueba 2

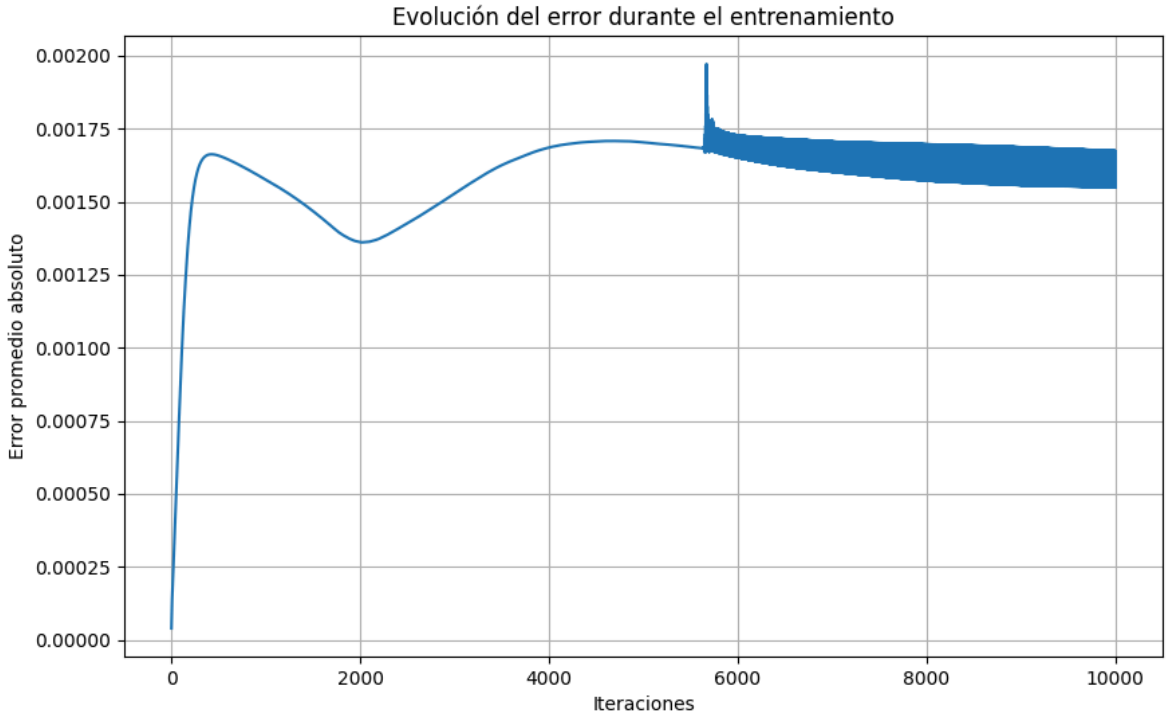

Figura 20: Comportamiento del error de la red durante el entrenamiento de la prueba 2

Fuente: Elaboración propia

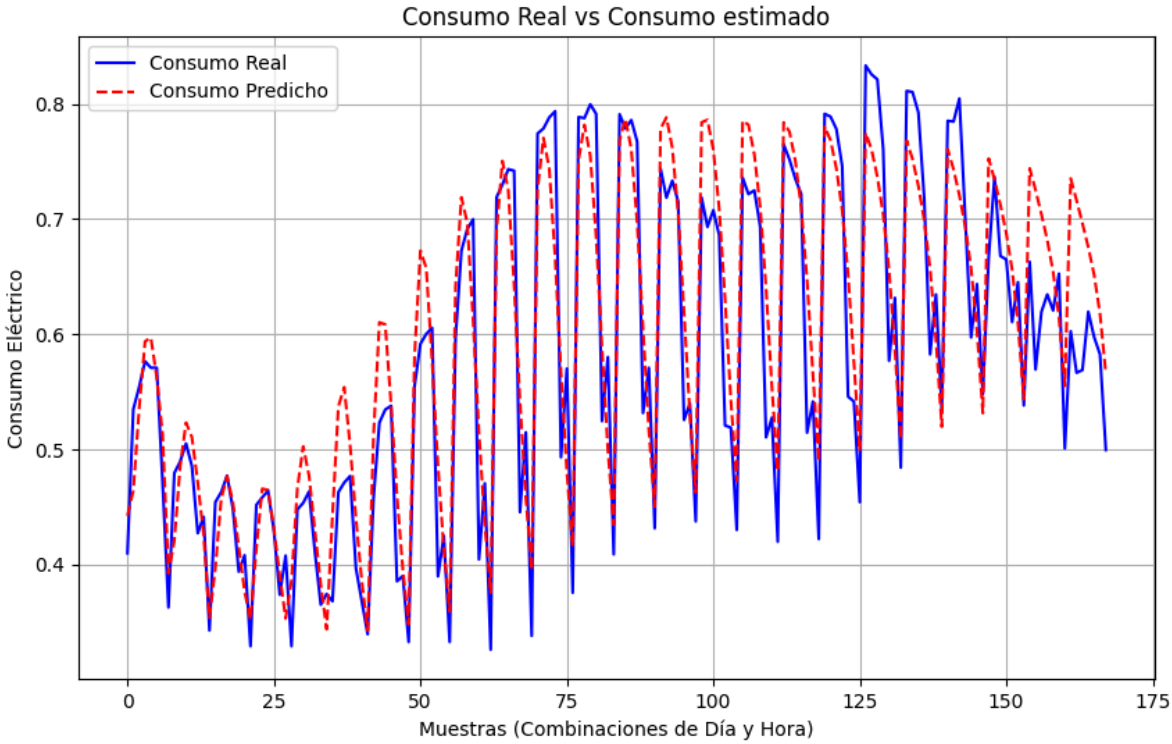

Figura 21: Consumo real vs estimado por la red en la prueba 2

Fuente: Elaboración propia

```
Ingrese un dia de la semana (0 a 6): 1
Ingrese la hora del día (1 a 24): 7
El consumo electrico en el dia 1 a las 7:00 es: 0.6726
```

Figura 22: Resultados por consola de la prueba 2

Fuente: Elaboración propia

Análisis de los resultados

Para la arquitectura A

Se puede ver que en la gráfica del error, hay una leve diferencia, pues el error en la prueba 1 empieza a oscilar antes de las 5000 iteraciones, mientras que en la prueba 2, esto se da a partir de las 5000 iteraciones, para luego ambas terminar a las 10000 de forma similar. Aunque ambas pruebas no hayan alcanzado un error de 0, por lo menos es un error bastante pequeño, por lo que la precisión en la respuesta es bastante acertada; sin embargo, por consola se puede notar cómo afectó este comportamiento en la respuesta, donde en la primera prueba que empezó a oscilar el error temprano, arrojó un valor de 0.6420, y para la segunda 0.7066, donde el valor objetivo es 0.7460. Esto permite decir que esta arquitectura puede presentar alta exactitud pero baja precisión.

Para la arquitectura B

Utilizando esta arquitectura se logra un mejor resultado, puesto que en ambas gráficas, la del error y la del consumo, tienen un comportamiento bastante similar, por lo que permite concluir que su precisión y exactitud es alta, aún sin ver el valor arrojado por consola, donde para la primera prueba, se obtiene 0,5533, y para la segunda 0.5602, donde el valor objetivo es 0.5480. Esta arquitectura es más confiable que la A, puesto que a pesar de que solo se han hecho 2 pruebas en cada una, para este contexto los resultados entre una y otra prueba no son tan alejados del valor teórico y entre estos mismos.

Para la arquitectura C

Esta arquitectura a diferencia de las anteriores dos, es bastante diferente en su comportamiento, puesto que en su consumo eléctrico se evidencia incoherencia y un error en los datos de consumo, cosa que en la gráfica que muestra el error, no se evidencia mucho en eje de las y, solamente se nota un distinto comportamiento en el momento que oscila entre finales y las 5000 iteraciones y la mitad de las 6000; por otro lado, el valor de salida entregado por consola es donde más se nota

el error en la exactitud, dónde la primera prueba arroja un valor de 0.6318 y para la segunda 0,6726, donde el valor objetivo es de 0.5233. Se puede decir que esta arquitectura es la que más falló, puesto que los valores arrojados tuvieron una precisión considerable entre las pruebas, pero alejadas del valor real (baja exactitud)

Conclusiones

Para concluir con el análisis, se puede decir que la red neuronal multicapa se implementó de forma satisfactoria, sin embargo, presenta errores en sus resultados dependiendo de la configuración que se le haga a la red, en cuanto a sus capas ocultas y cantidad de neuronas, lo cual hace recalcar la importancia de este punto al momento de diseñar una red en base al requerimiento que pueda tener un usuario, si requiere valores muy precisos o aproximados.

Por otro lado, la implementación de esta red podrá ser de gran utilidad para predecir el consumo eléctrico de una fecha y día determinada, lo cuál puede ayudar a situaciones como:

- Cuándo realizar tareas de mantenimiento al sistema eléctrico de la población, y que no afecte mucho a los individuos.
- Implementar medidas y regulaciones para disminuir el consumo eléctrico en fechas puntuales
- Si se planea realizar un evento de gran tamaño, saber establecer la hora de realización en un horario prudente y adecuado

Y cómo último, la funciona a partir de los datos ingresados como .txt suministrado por el material de trabajo, no se realizó una prueba con un archivos que contenga muchos más datos, pero esto puede ser una acción a realizar a futuro, lo cual sería de ayuda porque si se suministra datos de consumo mayores a un año, podría alimentar de mejor manera al perceptrón y así obtener mejores resultados.