```
# Importar las bibliotecas necesarias
!pip install ipywidgets tensorflow matplotlib numpy tensorflowjs

import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras import Input
import matplotlib.pyplot as plt
import seaborn as sns
from ipywidgets import widgets, Button
from IPython.display import display

# Definir los datos de entrada (2.2) Num viv/Num hogares
X = np.array([131, 139, 131, 137, 127, 126, 137, 110, 124, 134,
124, 121, 140, 160, 140, 122, 135, 135, 120, 141,
              115, 101, 127, 132, 114, 120, 142, 115, 120, 122,
121, 129, 123, 117])

# Definir los valores target
Y = np.array([  (21,13.04,75.60,1.39,9.48,0.15,1.37,0.05,0.26,-
2.73,18.56,8.12,9.49,4.67,1.00),
(24,15.74,13.00,6.95,7.99,1.68,11.88,2.31,2.11,26.14,54.45,3.43,9.7
6,5.32,3.03),
(34,11.53,11.05,1.37,8.03,0.68,2.48,0.45,0.80,26.81,77.15,1.54,13.3
5,6.46,2.38),
(16,11.68,9.46,1.92,9.57,0.25,1.24,0.11,0.77,-
11.92,81.02,4.13,9.77,5.72,1.85),
(27,7.83,53.95,1.15,8.49,0.10,0.86,1.26,0.28,-
0.23,41.12,3.76,8.25,4.02,0.83),
(14,18.34,44.91,13.56,10.15,1.64,3.50,0.02,2.57,-
25.55,19.55,6.56,18.60,9.71,5.01),
(24,3.48,13.15,0.29,9.40,0.05,0.42,1.13,0.16,-
4.22,83.85,0.34,7.85,4.34,0.78),
(26,13.42,0.00,14.59,7.92,9.41,0.45,0.30,2.69,-
52.57,40.69,5.85,34.22,14.99,14.17),
(20,16.64,36.76,10.14,9.24,1.60,10.55,0.20,1.37,-
13.41,29.67,6.21,10.08,4.96,4.60),
(19,0.94,64.50,1.32,10.43,0.26,2.50,0.11,0.52,-
5.33,24.54,0.35,9.23,4.85,2.39),
(38,13.66,67.11,1.22,8.12,0.26,2.24,0.63,0.31,-
17.08,25.77,2.50,7.97,3.69,1.24),
(15,24.37,6.32,16.46,10.06,6.54,6.35,1.48,5.86,-
2.93,33.55,8.36,25.87,13.16,6.69),
(19,4.83,9.93,0.48,9.57,0.06,0.50,0.82,0.11,-
31.76,87.15,1.69,10.98,6.18,0.58),
(15,5.50,10.65,7.30,11.20,2.65,8.38,0.15,4.72,51.87,55.07,3.10,15.9
5,9.41,3.57),
```

```
(37,1.96,32.45,7.02,8.62,2.65,7.05,0.01,2.31,-
26.54,36.58,0.17,19.50,11.49,3.45),
(18,2.50,57.48,1.69,10.46,0.37,2.81,0.25,0.56,46.11,33.16,0.97,12.1
1,5.65,1.72),
(31,4.65,66.76,1.95,6.52,0.27,1.08,0.69,0.39,-
31.62,22.67,1.22,8.31,4.26,1.73),
(18,11.13,9.21,11.71,9.11,3.18,9.02,0.69,4.71,-
49.11,39.29,4.09,14.80,8.91,5.92),
(14,3.65,73.20,4.20,9.70,0.66,3.91,0.15,1.55,-
0.94,10.19,0.14,12.15,5.58,3.33),
(11,10.43,30.56,1.43,9.09,0.30,4.16,1.04,0.53,-
13.74,55.84,0.90,4.52,2.69,3.70),
(22,15.06,22.17,4.26,7.49,1.46,4.66,1.41,1.31,22.66,54.55,2.71,16.2
1,6.94,5.08),
(32,28.86,1.74,16.40,8.40,6.57,7.58,10.44,8.30,1.73,33.26,8.28,22.6
6,6.53,5.34),
(19,24.38,47.44,2.29,8.80,0.52,3.58,1.24,0.72,55.07,38.81,6.20,10.2
9,4.56,2.07),
(15,10.70,51.71,4.20,9.06,0.82,3.17,0.00,1.37,-
40.76,34.25,7.85,14.14,7.94,1.51),
(28,44.57,4.02,6.83,8.53,3.74,8.92,1.84,3.64,-
4.41,53.92,15.88,18.59,7.87,6.14),
(16,10.39,18.59,2.62,9.26,0.71,6.38,2.19,1.26,-
3.44,62.44,3.22,7.50,3.69,4.17),
(18,15.50,20.34,11.99,8.07,3.67,9.93,3.16,7.02,-
33.04,29.38,4.41,18.73,11.79,6.49),
(32,39.27,1.78,2.88,8.44,1.39,5.32,3.97,1.01,14.32,82.25,7.46,15.99
,7.04,2.26),
(24,36.99,27.87,13.84,8.61,4.89,3.70,0.99,9.41,-
21.32,12.06,3.05,22.25,10.22,8.88),
(19,11.10,12.48,10.96,10.25,2.44,10.64,0.28,4.62,24.64,44.26,1.37,1
4.24,6.65,8.75),
(15,6.81,42.22,2.74,11.34,0.37,4.29,2.38,0.99,16.82,42.00,0.85,8.04
,3.62,2.57),
(19,81.33,28.14,8.15,9.57,5.87,5.28,2.62,4.07,4.39,30.70,10.49,26.1
6,13.42,5.73),
(14,32.79,45.17,8.52,9.55,0.00,5.73,7.35,3.07,-
33.37,26.37,12.09,14.44,7.53,3.00),
(13,41.30,30.47,3.71,10.22,0.70,2.60,3.80,0.99,11.63,53.49,15.22,16
.90,8.04,3.61)

])


# Normalizar los datos de entrada y salida
X_max, Y_max = np.max(X), np.max(Y)
X_normalized, Y_normalized = X / X_max, Y / Y_max
```

```
# Red neuronal ajustada
model = Sequential([
    Input(shape=(1,)),
    Dense(64, activation='relu',
kernel_regularizer=tf.keras.regularizers.l2(0.0001)),
    Dense(128, activation='relu',
kernel_regularizer=tf.keras.regularizers.l2(0.0001)),
    Dense(128, activation='relu',
kernel_regularizer=tf.keras.regularizers.l2(0.0001)),
    Dense(15, activation='linear')
])

# Compilación y entrenamiento con ajuste en tasa de aprendizaje y
optimizador Adam
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.00
01), loss='mean_squared_error')
historial = model.fit(X_normalized, Y_normalized, epochs=100,
verbose=1, validation_split=0.2)


# Gráfica de pérdida ajustada
plt.xlabel('#Época')
plt.ylabel('Pérdida')
plt.plot(historial.history['loss'], label='Entrenamiento')
plt.plot(historial.history['val_loss'], label='Validación')
plt.legend()
plt.show()

# Funciones de predicción y cálculo de error
def predecir(value):
    value_normalized = np.array([value]) / X_max
    prediction = model.predict(value_normalized)
    return prediction * Y_max

# Definir la función para calcular el error absoluto
def calcular_error(real_target, prediccion_target):
    return np.abs(real_target - prediccion_target)

# Obtener las predicciones y los valores reales
predicciones = model.predict(X_normalized) * Y_max  # Predicciones
escaladas a los valores originales
valores_reales = Y  # Matriz con los valores reales de Y

# Calcular el error absoluto entre cada valor real y predicción
errores = calcular_error(valores_reales, predicciones)
errores = np.abs(valores_reales - predicciones) # Calcular errores
y asignar a la variable 'errores'
```

```
# Gráfica de error para cada output
plt.figure(figsize=(12, 8))
for i in range(15):
    plt.subplot(3, 5, i + 1)
    plt.bar(range(len(errores)), errores[:, i], color='salmon')
    plt.xlabel('Muestra')
    plt.ylabel(f'Error Output {i+1}')
    plt.title(f'Error en Output {i+1}')
plt.tight_layout()
plt.show()

# Crear y mostrar tabla de predicción 1
def crear_tabla_4x4(value, prediction):
    tabla_4x4 = np.zeros((4, 4))
    tabla_4x4[1, 1] = value
    tabla_4x4[0, :] = prediction[0][:4]
    tabla_4x4[1, [0, 2, 3]] = prediction[0][4:7]
    tabla_4x4[2, :] = prediction[0][7:11]
    tabla_4x4[3, :] = prediction[0][11:]
    return tabla_4x4

names = [
    ["n0/n1", "INVMUN M", "% AGR", "% IND_COM_PUB_MIL_PRIV"],
    ["SALARIO m€", "viv/hog", "DPOB mhab/km2", "% RESD"],
    ["INVMA M€", "% V_D_O", "Δ POB14/21", "% NAT"],
    ["INVVU M€", "DPOBTRES m hab/km2", "COMPRES m viv/Km2res",
"%INFT"]
]

def mostrar_tabla_4x4(tabla_4x4, titulo="Tabla de Predicción 4x4"):
    plt.figure(figsize=(10, 10))
    sns.heatmap(tabla_4x4,
annot=[[f"{names[i][j]}\n{tabla_4x4[i][j]:.2f}" for j in range(4)]
for i in range(4)],
                fmt="", cmap='coolwarm', cbar=True, linewidths=1,
linecolor='black', xticklabels=[4, 3, 2, 1],
                yticklabels=[4, 3, 2, 1])
    plt.title(titulo)
    plt.show()

# Crear widgets
nuevo_valor_slider = widgets.FloatSlider(min=0, max=200, step=1,
value=131, description="Entrada 2.2:")
boton_actualizar = Button(description="Actualizar Predicción")



# Función para actualizar la predicción al hacer clic en el botón
def on_button_clicked(b):
```

```
    value = nuevo_valor_slider.value
    prediction = predecir(value)
    tabla_4x4 = crear_tabla_4x4(value, prediction)
    mostrar_tabla_4x4(tabla_4x4)
    mostrar_error(value, prediction)
    graficar_relaciones(value, prediction)

def mostrar_error(value, prediction):
    # Add your error display logic here
    print(f"Error for value {value}: {prediction}")  # Example:
Print the prediction as error

def graficar_relaciones(value, prediction):
    # Add your relationship plotting logic here
    print(f"Plotting relationships for value {value}:
{prediction}")  # Example: Print the prediction


# Conectar el botón y mostrar la interfaz
boton_actualizar.on_click(on_button_clicked)
display(nuevo_valor_slider, boton_actualizar)

# Predicción inicial
initial_prediction = predecir(nuevo_valor_slider.value)
tabla_4x4_inicial = crear_tabla_4x4(nuevo_valor_slider.value,
initial_prediction)
mostrar_tabla_4x4(tabla_4x4_inicial, titulo="Predicción Inicial con
Entrada 2.2")

# Gráfica de dispersión final
predicciones = model.predict(X_normalized) * Y_max
plt.figure(figsize=(10, 6))
for i in range(15):
    valores_reales_output = Y[:, i]
    predicciones_output = predicciones[:, i]
    plt.scatter(valores_reales_output, predicciones_output,
label=f'Output {i+1}', alpha=0.6, s=50)

# Línea de igualdad (y = x)
plt.plot([min(Y.flatten()), max(Y.flatten())], [min(Y.flatten()),
max(Y.flatten())], 'r--', label='Línea de Igualdad')
plt.xlabel('Valores Reales')
plt.ylabel('Valores Predichos')
plt.title('Dispersión de Valores Reales vs Predicciones para 15
Outputs')
plt.legend()
plt.show()
```