ACT3

September 11, 2024

0.0.1 ACTIVIDAD: AJUSTE DE REDES NEURONALES

A01285158 | Grace Aviance Silva Aróstegui

0.0.2 EJERCICIO 1

El conjunto de datos de *criminalidad* de Estados Unidos publicado en el año 1993 consiste de 51 registros para los que se tienen las siguientes variables:

- VR = crímenes violentos por cada 100000 habitantes
- MR = asesinatos por cada 100000 habitantes
- M = porcentaje de áreas metropolitanas
- W = porcentaje de gente blanca
- H = porcentaje de personas con preparatoria terminada
- P = porcentaje con ingresos por debajo del nivel de pobreza
- S = porcentaje de familias con solo un miembro adulto como tutor Para este conjunto de datos:

1. Evalúa con validación cruzada un modelo perceptrón multicapa para las variables que se te asignaron para este ejercicio.

2. Viendo los resultados de regresión, desarrolla una conclusión sobre los siguientes puntos:

A.¿Consideras que el modelo perceptrón multicapa es efectivo para modelar los datos del problema? ¿Por qué?

B.¿Qué modelo es mejor para los datos de criminalidad, el lineal o el perceptrón multicapa? ¿Por qué?

Nota: Las variables con las que vas a trabajar depende del último número de tu matrícula de acuerdo a la siguiente lista: 8 - Variable dependiente VR, variables independientes M, W, H, P y S

[81]:
```
# LIBRERIAS
import warnings
warnings.filterwarnings("ignore")
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPRegressor
from sklearn.model_selection import GridSearchCV, cross_val_predict
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import matplotlib.pyplot as plt
from sklearn.pipeline import Pipeline, make_pipeline
```

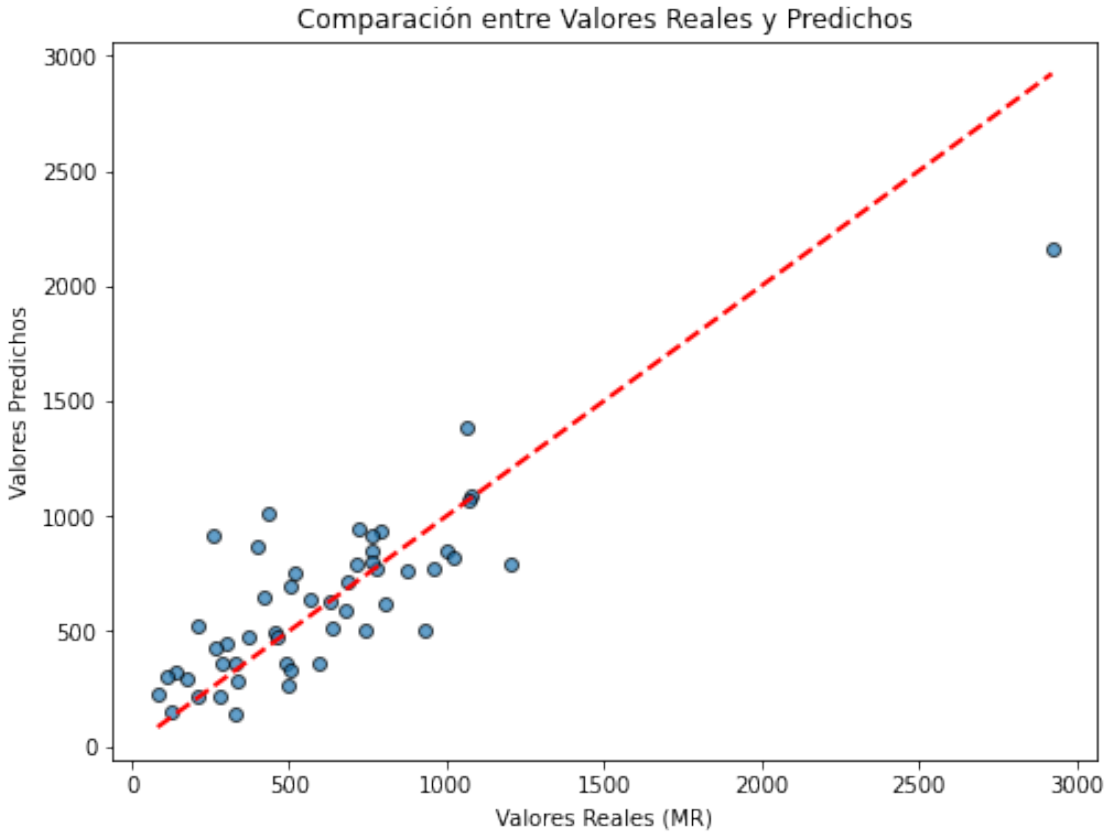
[2]:
```
df1 = pd.read_csv("crime_data.csv")
df1.shape
```

[2]:
```
(51, 8)
```

[3]:
```
# Definir las características y el objetivo
caracteristicas = df1.loc[:, ['M', 'W', 'H', 'P', 'S']] # Variables predictoras
objetivo = df1.loc[:, 'VR'] # Variable de respuesta

# Crear una tubería para el preprocesamiento y el modelo
modelo = Pipeline([
    ('normalizador', StandardScaler()),
    ('red_neuronal', MLPRegressor(max_iter=10000))
])

# Especificar el conjunto de hiperparámetros para la búsqueda
hiperparametros = {
    'red_neuronal__hidden_layer_sizes': [(50,), (100,), (50, 50), (100, 100)],
    'red_neuronal__activation': ['relu', 'tanh'],
    'red_neuronal__solver': ['adam', 'sgd'],
    'red_neuronal__alpha': [0.0001, 0.001, 0.01],
    'red_neuronal__learning_rate': ['constant', 'adaptive']
}

# Configurar la búsqueda de hiperparámetros con validación cruzada
busqueda = GridSearchCV(modelo, hiperparametros, cv=5,␣
 ↪scoring='neg_mean_squared_error')
busqueda.fit(caracteristicas, objetivo)

# Seleccionar el mejor modelo encontrado por la búsqueda de hiperparámetros
mejor_modelo = busqueda.best_estimator_

# Generar predicciones utilizando validación cruzada
predicciones = cross_val_predict(mejor_modelo, caracteristicas, objetivo, cv=5)

# Calcular las métricas de rendimiento
error_cuadratico_medio = mean_squared_error(objetivo, predicciones)
error_absoluto_medio = mean_absolute_error(objetivo, predicciones)
coeficiente_determinacion = r2_score(objetivo, predicciones)

# Mostrar los resultados
print("Mejores hiperparámetros encontrados:", busqueda.best_params_)
print("Error Cuadrático Medio (MSE):", error_cuadratico_medio)
print("Error Absoluto Medio (MAE):", error_absoluto_medio)
print("Coeficiente de Determinación (R²):", coeficiente_determinacion)

# Visualizar la relación entre valores reales y predichos
```

```
plt.figure(figsize=(8, 6))
plt.scatter(objetivo, predicciones, edgecolor='k', alpha=0.7)
plt.plot([objetivo.min(), objetivo.max()], [objetivo.min(), objetivo.max()],␣
 ↪'r--', lw=2)
plt.xlabel('Valores Reales (MR)')
plt.ylabel('Valores Predichos')
plt.title('Comparación entre Valores Reales y Predichos')
plt.show()
```

```
Mejores hiperparámetros encontrados: {'red_neuronal__activation': 'relu',
'red_neuronal__alpha': 0.001, 'red_neuronal__hidden_layer_sizes': (100,),
'red_neuronal__learning_rate': 'adaptive', 'red_neuronal__solver': 'adam'}
Error Cuadrático Medio (MSE): 58089.07354443814
Error Absoluto Medio (MAE): 177.01259785217997
Coeficiente de Determinación (R²): 0.6954771607687933
```



A.¿Consideras que el modelo perceptrón multicapa es efectivo para modelar los datos del problema? ¿Por qué?

No diría que es realmente efectivo ya que los errores son grandes y el coeficiente de determinación no es tan alto como para decir que es efectivo. Sin embargo tampoco esta tan mal, si da una noción de los datos. Al ser R^2 de 0.69 pudiera considerarse un bueno modelo pero no lo suficiente para

etiquetarlo como efectivo.

B.¿Qué modelo es mejor para los datos de criminalidad, el lineal o el perceptrón multicapa? ¿Por qué?

Pensaría que no habría mucha diferencia entre el modelado, pero tengo la idea de que con el modelo lineal tomaría menos tiempo computacional

0.0.3 EJERCICIO 2

En este ejercicio trabajarás con datos que vienen de un experimento en el que se midió actividad muscular con la técnica de la Electromiografía en el brazo derecho de varios participantes cuando éstos realizaban un movimiento con la mano entre siete posible (Flexionar hacia arriba, Flexionar hacia abajo, Cerrar la mano, Estirar la mano, Abrir la mano, Coger un objeto, No moverse).

Al igual que en el ejercicio anterior, los datos se cargan con la función *loadtxt de numpy* (https://numpy.org/doc/stable/reference/generated/numpy.loadtxt.html).

A su vez, la primera columna corresponde a la clase (1, 2, 3, 4, 5, 6, y 7), la segunda columna se ignora, y el resto de las columnas indican las variables que se calcularon de la respuesta muscular. El archivo de datos con el que trabajarás depende de tu matrícula.

Para este conjunto de datos:

1. Evalúa un modelo perceptrón multicapa con validación cruzada utilizando al menos 5 capas de 20 neuronas.

2. Evalúa un modelo perceptrón multicapa con validación cruzada, pero encontrando el número óptimo de capas y neuronas de la red.

3. Prepara el modelo perceptrón multicapa:

A. Opten los hiperparámetros óptimos de capas y neuronas de la red.

B. Con los hiperparámetros óptimos, ajusta el modelo con todos los datos.

4. Contesta lo siguientes:

A. ¿Observas alguna mejora importante al optimizar el tamaño de la red? ¿Es el resultado que esperabas? Argumenta tu respuesta.

B. ¿Qué inconvenientes hay al encontrar el tamaño óptimo de la red? ¿Por qué?

[91]:
```
from sklearn.neural_network import MLPClassifier, MLPRegressor
from sklearn.model_selection import KFold, StratifiedKFold, cross_val_score,␣
 ↪cross_val_predict, train_test_split
from sklearn.pipeline import Pipeline, make_pipeline
from sklearn.model_selection import KFold
from keras import Sequential
from keras.layers import Dense
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.linear_model import LinearRegression
from sklearn.metrics import classification_report
from keras.utils import to_categorical
```

```
import numpy as np
from sklearn.preprocessing import MinMaxScaler
```

[95]:
```
#df2 = pd.read_csv('M_3.txt', sep='\t', header=None)
df2=np.loadtxt('M_3.txt')
#X = df2.iloc[:, 2:-1]
X = df2[:,2:]
#y = df2.iloc[:, 0]
y = df2[:,0]
```

[79]:
```
# EVALUACION DE PERCEPTRON MULTICAPA

# Modelo MLP con 6 capas ocultas de 15 neuronas cada una
modelo_mlp = MLPClassifier(hidden_layer_sizes=[15] * 6, max_iter=1000,␣
 ↪random_state=42)

# Escalado de las características
pipeline = make_pipeline(StandardScaler(), modelo_mlp)

# Cross validation para evaluar el rendimiento del modelo
resultado_validacion = cross_val_score(pipeline, X, y, cv=5, scoring='accuracy')

# Mostrar la exactitud promedio
print(f'Exactitud promedio: {resultado_validacion.mean()}')
```

```
Exactitud promedio: 0.9015873015873016
```

[86]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPClassifier
# Crear el pipeline con el escalador y el modelo
pipeline = make_pipeline(StandardScaler(), MLPClassifier(max_iter=1000,␣
 ↪random_state=42))

# Rango de hiperparámetros a buscar
param_grid = {
    'mlpclassifier__hidden_layer_sizes': [
        (10, 10, 10, 10, 10, 10),  # 6 capas de 10 neuronas
        (20, 20, 20, 20, 20, 20),  # 6 capas de 20 neuronas
        (20, 20, 20, 20, 20),      # 5 capas de 20 neuronas
        (30, 30, 30, 30, 30),      # 5 capas de 30 neuronas
        (60, 60, 60),              # 3 capas de 60 neuronas
        (80, 80, 80),              # 3 capas de 80 neuronas
        (80, 80),                  # 2 capas de 80 neuronas
        (100, 100),                # 2 capas de 100 neuronas
        (100,)                     # 1 capa de 100 neuronas
```

```
    ]
}

# Crear la búsqueda de hiperparámetros
#grid_search = GridSearchCV(pipeline, param_grid, cv=5, scoring='accuracy',␣
 ↪n_jobs=-1)

# Ejecutar la búsqueda de hiperparámetros
grid_search.fit(X, y)

print(f'Mejor configuración de capas y neuronas: {grid_search.best_params_}')
```

```
Mejor configuración de capas y neuronas: {'mlpclassifier__hidden_layer_sizes':
(100,)}
```

[96]:
```
x = X

features = x.shape[1]
features
num_classes = 7
y = to_categorical(y - 1, num_classes=num_classes)  # Restar 1 si las clases␣
 ↪están en el rango [1, 7]

# Definir el número de características
features = x.shape[1]

n_folds = 5
kfold = KFold(n_splits=n_folds, shuffle=True)
cv_y_test = []
cv_y_pred = []
it = 0

for train_index, test_index in kfold.split(x, y):
    it += 1
    print("it: ", it)

    x_train = x[train_index, :]
    y_train = y[train_index]

    x_test = x[test_index, :]
    y_test = y[test_index]

    # Fase de entrenamiento
    # Definir el modelo MLP
    mlp_model = Sequential()
    mlp_model.add(Dense(20, activation='relu', input_dim=features))
    mlp_model.add(Dense(20, activation='relu'))
```

```
    mlp_model.add(Dense(20, activation='relu'))
    mlp_model.add(Dense(20, activation='relu'))
    mlp_model.add(Dense(20, activation='relu'))
    mlp_model.add(Dense(num_classes, activation='softmax'))  # Capa de salida

    # Compilar el modelo
    mlp_model.compile(optimizer='adam', loss='categorical_crossentropy',␣
 ↪metrics=['accuracy'])

    # Entrenar el modelo
    mlp_model.fit(x_train, y_train, epochs=500, batch_size=32, verbose=0)

    # Predecir en los datos de prueba
    y_pred_proba = mlp_model.predict(x_test)
    y_pred = np.argmax(y_pred_proba, axis=1)
    y_test_labels = np.argmax(y_test, axis=1)

    # Agregar los valores verdaderos y predicciones
    cv_y_test.append(y_test_labels)
    cv_y_pred.append(y_pred)

# Convertir listas de arrays a un solo array para la evaluación final
cv_y_test = np.concatenate(cv_y_test)
cv_y_pred = np.concatenate(cv_y_pred)

# Imprimir el reporte de clasificación
print(classification_report(cv_y_test, cv_y_pred))
```

```
it:  1
4/4 [==============================] - 0s 513us/step
it:  2
4/4 [==============================] - 0s 6ms/step
it:  3
4/4 [==============================] - 0s 0s/step
it:  4
4/4 [==============================] - 0s 0s/step
it:  5
4/4 [==============================] - 0s 0s/step
              precision    recall  f1-score   support

           0       0.98      0.99      0.98        90
           1       0.97      0.94      0.96        90
           2       0.95      0.93      0.94        90
           3       0.99      1.00      0.99        90
           4       0.99      0.98      0.98        90
           5       0.92      0.90      0.91        90
           6       0.94      0.99      0.96        90
```

```
    accuracy                           0.96       630
   macro avg       0.96      0.96      0.96       630
weighted avg       0.96      0.96      0.96       630
```

[101]:
```
# 2. EVALUACIÓN DE PERCEPTRÓN MULTICAPA encontrando el num óptimo de capas y␣
 ↪neuronas

x=df2[:,2:]
y=df2[:,0]

x = MinMaxScaler().fit_transform(x)
features = x.shape[1]
num_classes = 7
y = to_categorical(y - 1, num_classes=num_classes)

# Configuraciones de capas para el MLP
configuraciones_capas = [
    [50, 20],               # 2 capas con diferente número de neuronas
    [20, 20, 20, 20, 20],   # 5 capas con 20 neuronas en cada una
    [40, 10, 20],           # 3 capas con diferente número de neuronas
    [30, 30, 30, 30]        # 4 capas con 25 neuronas en cada una
]

n_folds = 5
best_config = None
best_accuracy = 0

for config in configuraciones_capas:
    print(f"Evaluando configuración: {config}")
    cv_y_test = []
    cv_y_pred = []

    kfold = KFold(n_splits=n_folds, shuffle=True)

    for train_index, test_index in kfold.split(x, y):
        x_train = x[train_index, :]
        y_train = y[train_index]

        x_test = x[test_index, :]
        y_test = y[test_index]

        # Fase de entrenamiento
        # Definir el modelo MLP
        mlp_model = Sequential()
        mlp_model.add(Dense(config[0], activation='relu', input_dim=features))
```

```
        for units in config[1:]:
            mlp_model.add(Dense(units, activation='relu'))

        mlp_model.add(Dense(num_classes, activation='softmax'))  # Capa de␣
↪salida

        # Compilar el modelo
        mlp_model.compile(optimizer='adam', loss='categorical_crossentropy',␣
↪metrics=['accuracy'])

        # Entrenar el modelo
        mlp_model.fit(x_train, y_train, epochs=500, batch_size=32, verbose=0)

        # Predecir en los datos de prueba
        y_pred_proba = mlp_model.predict(x_test)
        y_pred = np.argmax(y_pred_proba, axis=1)
        y_test_labels = np.argmax(y_test, axis=1)

        # Agregar los valores verdaderos y predicciones
        cv_y_test.append(y_test_labels)
        cv_y_pred.append(y_pred)

    # Convertir listas de arrays a un solo array para la evaluación final
    cv_y_test = np.concatenate(cv_y_test)
    cv_y_pred = np.concatenate(cv_y_pred)

    # Evaluar el rendimiento
    accuracy = np.mean(cv_y_pred == cv_y_test)
    print(f"Precisión para configuración {config}: {accuracy}")

    # Guardar la mejor configuración
    if accuracy > best_accuracy:
        best_accuracy = accuracy
        best_config = config

print(f"La mejor configuración es {best_config} con una precisión de␣
↪{best_accuracy}")
```

```
Evaluando configuración: [50, 20]
4/4 [==============================] - 0s 5ms/step
4/4 [==============================] - 0s 0s/step
4/4 [==============================] - 0s 0s/step
4/4 [==============================] - 0s 0s/step
4/4 [==============================] - 0s 6ms/step
Precisión para configuración [50, 20]: 0.9666666666666667
Evaluando configuración: [20, 20, 20, 20, 20]
4/4 [==============================] - 0s 0s/step
```

```
4/4 [==============================] - 0s 0s/step
4/4 [==============================] - 0s 6ms/step
4/4 [==============================] - 0s 0s/step
4/4 [==============================] - 0s 1ms/step
Precisión para configuración [20, 20, 20, 20, 20]: 0.9380952380952381
Evaluando configuración: [40, 10, 20]
4/4 [==============================] - 0s 598us/step
4/4 [==============================] - 0s 5ms/step
4/4 [==============================] - 0s 5ms/step
4/4 [==============================] - 0s 0s/step
4/4 [==============================] - 0s 5ms/step
Precisión para configuración [40, 10, 20]: 0.9587301587301588
Evaluando configuración: [30, 30, 30, 30]
4/4 [==============================] - 0s 5ms/step
4/4 [==============================] - 0s 5ms/step
4/4 [==============================] - 0s 0s/step
4/4 [==============================] - 0s 0s/step
4/4 [==============================] - 0s 2ms/step
Precisión para configuración [30, 30, 30, 30]: 0.9619047619047619
La mejor configuración es [50, 20] con una precisión de 0.9666666666666667
```

[102]:
```python
# 3. PERCEPTRON MULTICAPA

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import RMSprop
from sklearn.metrics import classification_report
import numpy as np


# Crear el modelo MLP
mlp_model = Sequential()
mlp_model.add(Dense(units=best_config[0], activation='relu',␣
 ↪input_shape=(x_train.shape[1],)))

# Agregar capas adicionales según la mejor configuración
for units in best_config[1:]:
    mlp_model.add(Dense(units=units, activation='relu'))

# Capa de salida
mlp_model.add(Dense(units=num_classes, activation='softmax'))

# Compilar el modelo con un optimizador diferente y una función de pérdida␣
 ↪adecuada
mlp_model.compile(optimizer=RMSprop(learning_rate=0.001),␣
 ↪loss='categorical_crossentropy', metrics=['accuracy'])
```

```python
# Entrenar el modelo
mlp_model.fit(x_train, y_train, epochs=300, batch_size=64, verbose=1)  # Cambié␣
↪los valores de épocas y tamaño de lote

# Predecir en los datos de entrenamiento
Y_pred_proba = mlp_model.predict(x_train)
Y_pred = np.argmax(Y_pred_proba, axis=1)
Y_true = np.argmax(y_train, axis=1)

# Imprimir el reporte de clasificación
print(classification_report(Y_true, Y_pred))
```

```
Epoch 1/300
8/8 [==============================] - 0s 2ms/step - loss: 1.9868 - accuracy:
0.1687
Epoch 2/300
8/8 [==============================] - 0s 4ms/step - loss: 1.8026 - accuracy:
0.2520
Epoch 3/300
8/8 [==============================] - 0s 1ms/step - loss: 1.6035 - accuracy:
0.4881
Epoch 4/300
8/8 [==============================] - 0s 3ms/step - loss: 1.5367 - accuracy:
0.5060
Epoch 5/300
8/8 [==============================] - 0s 2ms/step - loss: 1.3768 - accuracy:
0.5873
Epoch 6/300
8/8 [==============================] - 0s 1ms/step - loss: 1.2870 - accuracy:
0.5794
Epoch 7/300
8/8 [==============================] - 0s 0s/step - loss: 1.1963 - accuracy:
0.6667
Epoch 8/300
8/8 [==============================] - 0s 208us/step - loss: 1.0669 - accuracy:
0.7421
Epoch 9/300
8/8 [==============================] - 0s 8ms/step - loss: 1.0557 - accuracy:
0.6310
Epoch 10/300
8/8 [==============================] - 0s 2ms/step - loss: 0.9682 - accuracy:
0.7123
Epoch 11/300
8/8 [==============================] - 0s 2ms/step - loss: 0.8796 - accuracy:
0.7381
Epoch 12/300
8/8 [==============================] - 0s 2ms/step - loss: 0.8497 - accuracy:
0.7500
```

```
Epoch 13/300
8/8 [==============================] - 0s 2ms/step - loss: 0.8066 - accuracy:
0.7579
Epoch 14/300
8/8 [==============================] - 0s 3ms/step - loss: 0.7326 - accuracy:
0.8214
Epoch 15/300
8/8 [==============================] - 0s 2ms/step - loss: 0.7703 - accuracy:
0.7599
Epoch 16/300
8/8 [==============================] - 0s 2ms/step - loss: 0.6694 - accuracy:
0.8333
Epoch 17/300
8/8 [==============================] - 0s 2ms/step - loss: 0.6687 - accuracy:
0.8095
Epoch 18/300
8/8 [==============================] - 0s 2ms/step - loss: 0.6510 - accuracy:
0.7718
Epoch 19/300
8/8 [==============================] - 0s 2ms/step - loss: 0.5990 - accuracy:
0.8313
Epoch 20/300
8/8 [==============================] - 0s 3ms/step - loss: 0.5579 - accuracy:
0.8651
Epoch 21/300
8/8 [==============================] - 0s 2ms/step - loss: 0.5644 - accuracy:
0.8373
Epoch 22/300
8/8 [==============================] - 0s 2ms/step - loss: 0.5399 - accuracy:
0.8611
Epoch 23/300
8/8 [==============================] - 0s 3ms/step - loss: 0.5519 - accuracy:
0.8353
Epoch 24/300
8/8 [==============================] - 0s 2ms/step - loss: 0.4961 - accuracy:
0.8710
Epoch 25/300
8/8 [==============================] - 0s 1ms/step - loss: 0.5051 - accuracy:
0.8333
Epoch 26/300
8/8 [==============================] - 0s 3ms/step - loss: 0.4960 - accuracy:
0.8591
Epoch 27/300
8/8 [==============================] - 0s 3ms/step - loss: 0.4178 - accuracy:
0.8948
Epoch 28/300
8/8 [==============================] - 0s 776us/step - loss: 0.4156 - accuracy:
0.9127
```

```
Epoch 29/300
8/8 [==============================] - 0s 1ms/step - loss: 0.4479 - accuracy:
0.8611
Epoch 30/300
8/8 [==============================] - 0s 2ms/step - loss: 0.4268 - accuracy:
0.8671
Epoch 31/300
8/8 [==============================] - 0s 221us/step - loss: 0.3729 - accuracy:
0.9246
Epoch 32/300
8/8 [==============================] - 0s 4ms/step - loss: 0.4357 - accuracy:
0.8492
Epoch 33/300
8/8 [==============================] - 0s 651us/step - loss: 0.3460 - accuracy:
0.9246
Epoch 34/300
8/8 [==============================] - 0s 3ms/step - loss: 0.4021 - accuracy:
0.8631
Epoch 35/300
8/8 [==============================] - 0s 928us/step - loss: 0.3613 - accuracy:
0.9048
Epoch 36/300
8/8 [==============================] - 0s 3ms/step - loss: 0.3114 - accuracy:
0.9563
Epoch 37/300
8/8 [==============================] - 0s 1ms/step - loss: 0.3668 - accuracy:
0.9008
Epoch 38/300
8/8 [==============================] - 0s 4ms/step - loss: 0.3329 - accuracy:
0.9048
Epoch 39/300
8/8 [==============================] - 0s 3ms/step - loss: 0.3499 - accuracy:
0.8770
Epoch 40/300
8/8 [==============================] - 0s 3ms/step - loss: 0.3074 - accuracy:
0.9286
Epoch 41/300
8/8 [==============================] - 0s 2ms/step - loss: 0.3500 - accuracy:
0.8909
Epoch 42/300
8/8 [==============================] - 0s 2ms/step - loss: 0.2781 - accuracy:
0.9385
Epoch 43/300
8/8 [==============================] - 0s 1ms/step - loss: 0.2977 - accuracy:
0.9147
Epoch 44/300
8/8 [==============================] - 0s 3ms/step - loss: 0.2747 - accuracy:
0.9365
```

```
Epoch 45/300
8/8 [==============================] - 0s 2ms/step - loss: 0.2818 - accuracy:
0.9187
Epoch 46/300
8/8 [==============================] - 0s 3ms/step - loss: 0.2684 - accuracy:
0.9405
Epoch 47/300
8/8 [==============================] - 0s 2ms/step - loss: 0.3031 - accuracy:
0.9028
Epoch 48/300
8/8 [==============================] - 0s 2ms/step - loss: 0.2611 - accuracy:
0.9266
Epoch 49/300
8/8 [==============================] - 0s 2ms/step - loss: 0.2848 - accuracy:
0.9206
Epoch 50/300
8/8 [==============================] - 0s 3ms/step - loss: 0.2412 - accuracy:
0.9385
Epoch 51/300
8/8 [==============================] - 0s 2ms/step - loss: 0.2748 - accuracy:
0.9187
Epoch 52/300
8/8 [==============================] - 0s 1ms/step - loss: 0.2795 - accuracy:
0.9107
Epoch 53/300
8/8 [==============================] - 0s 1ms/step - loss: 0.2168 - accuracy:
0.9583
Epoch 54/300
8/8 [==============================] - 0s 2ms/step - loss: 0.2971 - accuracy:
0.8988
Epoch 55/300
8/8 [==============================] - 0s 3ms/step - loss: 0.2399 - accuracy:
0.9365
Epoch 56/300
8/8 [==============================] - 0s 4ms/step - loss: 0.2185 - accuracy:
0.9484
Epoch 57/300
8/8 [==============================] - 0s 542us/step - loss: 0.2848 - accuracy:
0.8988
Epoch 58/300
8/8 [==============================] - 0s 2ms/step - loss: 0.2049 - accuracy:
0.9583
Epoch 59/300
8/8 [==============================] - 0s 3ms/step - loss: 0.2202 - accuracy:
0.9365
Epoch 60/300
8/8 [==============================] - 0s 4ms/step - loss: 0.1952 - accuracy:
0.9524
```

```
Epoch 61/300
8/8 [==============================] - 0s 3ms/step - loss: 0.2253 - accuracy:
0.9306
Epoch 62/300
8/8 [==============================] - 0s 2ms/step - loss: 0.2257 - accuracy:
0.9385
Epoch 63/300
8/8 [==============================] - 0s 3ms/step - loss: 0.2019 - accuracy:
0.9385
Epoch 64/300
8/8 [==============================] - 0s 1ms/step - loss: 0.1807 - accuracy:
0.9623
Epoch 65/300
8/8 [==============================] - 0s 2ms/step - loss: 0.2135 - accuracy:
0.9365
Epoch 66/300
8/8 [==============================] - 0s 2ms/step - loss: 0.1771 - accuracy:
0.9603
Epoch 67/300
8/8 [==============================] - 0s 2ms/step - loss: 0.2106 - accuracy:
0.9425
Epoch 68/300
8/8 [==============================] - 0s 3ms/step - loss: 0.1717 - accuracy:
0.9563
Epoch 69/300
8/8 [==============================] - 0s 1ms/step - loss: 0.1668 - accuracy:
0.9544
Epoch 70/300
8/8 [==============================] - 0s 2ms/step - loss: 0.1917 - accuracy:
0.9504
Epoch 71/300
8/8 [==============================] - 0s 2ms/step - loss: 0.1785 - accuracy:
0.9544
Epoch 72/300
8/8 [==============================] - 0s 1ms/step - loss: 0.1899 - accuracy:
0.9444
Epoch 73/300
8/8 [==============================] - 0s 2ms/step - loss: 0.1588 - accuracy:
0.9663
Epoch 74/300
8/8 [==============================] - 0s 2ms/step - loss: 0.2054 - accuracy:
0.9365
Epoch 75/300
8/8 [==============================] - 0s 2ms/step - loss: 0.1811 - accuracy:
0.9385
Epoch 76/300
8/8 [==============================] - 0s 1ms/step - loss: 0.1435 - accuracy:
0.9722
```

```
Epoch 77/300
8/8 [==============================] - 0s 2ms/step - loss: 0.2159 - accuracy:
0.9226
Epoch 78/300
8/8 [==============================] - 0s 1ms/step - loss: 0.1398 - accuracy:
0.9663
Epoch 79/300
8/8 [==============================] - 0s 1ms/step - loss: 0.1426 - accuracy:
0.9643
Epoch 80/300
8/8 [==============================] - 0s 1ms/step - loss: 0.1938 - accuracy:
0.9405
Epoch 81/300
8/8 [==============================] - 0s 1ms/step - loss: 0.1463 - accuracy:
0.9683
Epoch 82/300
8/8 [==============================] - 0s 2ms/step - loss: 0.1382 - accuracy:
0.9722
Epoch 83/300
8/8 [==============================] - 0s 3ms/step - loss: 0.1501 - accuracy:
0.9524
Epoch 84/300
8/8 [==============================] - 0s 2ms/step - loss: 0.1454 - accuracy:
0.9663
Epoch 85/300
8/8 [==============================] - 0s 2ms/step - loss: 0.1438 - accuracy:
0.9623
Epoch 86/300
8/8 [==============================] - 0s 3ms/step - loss: 0.1871 - accuracy:
0.9405
Epoch 87/300
8/8 [==============================] - 0s 812us/step - loss: 0.1246 - accuracy:
0.9742
Epoch 88/300
8/8 [==============================] - 0s 3ms/step - loss: 0.1325 - accuracy:
0.9702
Epoch 89/300
8/8 [==============================] - 0s 2ms/step - loss: 0.1684 - accuracy:
0.9504
Epoch 90/300
8/8 [==============================] - 0s 2ms/step - loss: 0.1227 - accuracy:
0.9722
Epoch 91/300
8/8 [==============================] - 0s 2ms/step - loss: 0.1244 - accuracy:
0.9623
Epoch 92/300
8/8 [==============================] - 0s 2ms/step - loss: 0.1643 - accuracy:
0.9544
```

```
Epoch 93/300
8/8 [==============================] - 0s 838us/step - loss: 0.1691 - accuracy:
0.9385
Epoch 94/300
8/8 [==============================] - 0s 2ms/step - loss: 0.1289 - accuracy:
0.9603
Epoch 95/300
8/8 [==============================] - 0s 5ms/step - loss: 0.1077 - accuracy:
0.9802
Epoch 96/300
8/8 [==============================] - 0s 2ms/step - loss: 0.1435 - accuracy:
0.9563
Epoch 97/300
8/8 [==============================] - 0s 5ms/step - loss: 0.1035 - accuracy:
0.9802
Epoch 98/300
8/8 [==============================] - 0s 379us/step - loss: 0.1356 - accuracy:
0.9583
Epoch 99/300
8/8 [==============================] - 0s 5ms/step - loss: 0.1083 - accuracy:
0.9762
Epoch 100/300
8/8 [==============================] - 0s 850us/step - loss: 0.1397 - accuracy:
0.9603
Epoch 101/300
8/8 [==============================] - 0s 5ms/step - loss: 0.0930 - accuracy:
0.9821
Epoch 102/300
8/8 [==============================] - 0s 4ms/step - loss: 0.1409 - accuracy:
0.9563
Epoch 103/300
8/8 [==============================] - 0s 2ms/step - loss: 0.1148 - accuracy:
0.9762
Epoch 104/300
8/8 [==============================] - 0s 2ms/step - loss: 0.1150 - accuracy:
0.9623
Epoch 105/300
8/8 [==============================] - 0s 6ms/step - loss: 0.1583 - accuracy:
0.9504
Epoch 106/300
8/8 [==============================] - 0s 2ms/step - loss: 0.0970 - accuracy:
0.9683
Epoch 107/300
8/8 [==============================] - 0s 2ms/step - loss: 0.1612 - accuracy:
0.9365
Epoch 108/300
8/8 [==============================] - 0s 3ms/step - loss: 0.1015 - accuracy:
0.9722
```

```
Epoch 109/300
8/8 [==============================] - 0s 2ms/step - loss: 0.0996 - accuracy:
0.9762
Epoch 110/300
8/8 [==============================] - 0s 3ms/step - loss: 0.0979 - accuracy:
0.9722
Epoch 111/300
8/8 [==============================] - 0s 3ms/step - loss: 0.0857 - accuracy:
0.9762
Epoch 112/300
8/8 [==============================] - 0s 3ms/step - loss: 0.1292 - accuracy:
0.9504
Epoch 113/300
8/8 [==============================] - 0s 2ms/step - loss: 0.0900 - accuracy:
0.9782
Epoch 114/300
8/8 [==============================] - 0s 3ms/step - loss: 0.0849 - accuracy:
0.9802
Epoch 115/300
8/8 [==============================] - 0s 2ms/step - loss: 0.1445 - accuracy:
0.9425
Epoch 116/300
8/8 [==============================] - 0s 2ms/step - loss: 0.0839 - accuracy:
0.9821
Epoch 117/300
8/8 [==============================] - 0s 3ms/step - loss: 0.0814 - accuracy:
0.9782
Epoch 118/300
8/8 [==============================] - 0s 4ms/step - loss: 0.1011 - accuracy:
0.9722
Epoch 119/300
8/8 [==============================] - 0s 2ms/step - loss: 0.1274 - accuracy:
0.9663
Epoch 120/300
8/8 [==============================] - 0s 5ms/step - loss: 0.0888 - accuracy:
0.9702
Epoch 121/300
8/8 [==============================] - 0s 2ms/step - loss: 0.0935 - accuracy:
0.9762
Epoch 122/300
8/8 [==============================] - 0s 2ms/step - loss: 0.1137 - accuracy:
0.9603
Epoch 123/300
8/8 [==============================] - 0s 2ms/step - loss: 0.0881 - accuracy:
0.9762
Epoch 124/300
8/8 [==============================] - 0s 2ms/step - loss: 0.0849 - accuracy:
0.9782
```

```
Epoch 125/300
8/8 [==============================] - 0s 1ms/step - loss: 0.0865 - accuracy:
0.9722
Epoch 126/300
8/8 [==============================] - 0s 4ms/step - loss: 0.0836 - accuracy:
0.9782
Epoch 127/300
8/8 [==============================] - 0s 2ms/step - loss: 0.1071 - accuracy:
0.9603
Epoch 128/300
8/8 [==============================] - 0s 2ms/step - loss: 0.0751 - accuracy:
0.9841
Epoch 129/300
8/8 [==============================] - 0s 2ms/step - loss: 0.0658 - accuracy:
0.9841
Epoch 130/300
8/8 [==============================] - 0s 1ms/step - loss: 0.1227 - accuracy:
0.9484
Epoch 131/300
8/8 [==============================] - 0s 4ms/step - loss: 0.0628 - accuracy:
0.9861
Epoch 132/300
8/8 [==============================] - 0s 2ms/step - loss: 0.1374 - accuracy:
0.9444
Epoch 133/300
8/8 [==============================] - 0s 6ms/step - loss: 0.0688 - accuracy:
0.9802
Epoch 134/300
8/8 [==============================] - 0s 5ms/step - loss: 0.0595 - accuracy:
0.9901
Epoch 135/300
8/8 [==============================] - 0s 3ms/step - loss: 0.1067 - accuracy:
0.9623
Epoch 136/300
8/8 [==============================] - 0s 4ms/step - loss: 0.0606 - accuracy:
0.9841
Epoch 137/300
8/8 [==============================] - 0s 3ms/step - loss: 0.0792 - accuracy:
0.9841
Epoch 138/300
8/8 [==============================] - 0s 3ms/step - loss: 0.0597 - accuracy:
0.9881
Epoch 139/300
8/8 [==============================] - 0s 2ms/step - loss: 0.1220 - accuracy:
0.9603
Epoch 140/300
8/8 [==============================] - 0s 2ms/step - loss: 0.0530 - accuracy:
0.9901
```

```
Epoch 141/300
8/8 [==============================] - 0s 2ms/step - loss: 0.0710 - accuracy:
0.9821
Epoch 142/300
8/8 [==============================] - 0s 3ms/step - loss: 0.0775 - accuracy:
0.9742
Epoch 143/300
8/8 [==============================] - 0s 3ms/step - loss: 0.0638 - accuracy:
0.9861
Epoch 144/300
8/8 [==============================] - 0s 4ms/step - loss: 0.1124 - accuracy:
0.9544
Epoch 145/300
8/8 [==============================] - 0s 2ms/step - loss: 0.0547 - accuracy:
0.9881
Epoch 146/300
8/8 [==============================] - 0s 3ms/step - loss: 0.0688 - accuracy:
0.9742
Epoch 147/300
8/8 [==============================] - 0s 2ms/step - loss: 0.0629 - accuracy:
0.9821
Epoch 148/300
8/8 [==============================] - 0s 3ms/step - loss: 0.0965 - accuracy:
0.9702
Epoch 149/300
8/8 [==============================] - 0s 2ms/step - loss: 0.0538 - accuracy:
0.9901
Epoch 150/300
8/8 [==============================] - 0s 2ms/step - loss: 0.0615 - accuracy:
0.9802
Epoch 151/300
8/8 [==============================] - 0s 0s/step - loss: 0.0689 - accuracy:
0.9821
Epoch 152/300
8/8 [==============================] - 0s 3ms/step - loss: 0.0521 - accuracy:
0.9881
Epoch 153/300
8/8 [==============================] - 0s 4ms/step - loss: 0.0742 - accuracy:
0.9782
Epoch 154/300
8/8 [==============================] - 0s 2ms/step - loss: 0.0692 - accuracy:
0.9782
Epoch 155/300
8/8 [==============================] - 0s 4ms/step - loss: 0.0420 - accuracy:
0.9940
Epoch 156/300
8/8 [==============================] - 0s 3ms/step - loss: 0.0505 - accuracy:
0.9861
```

```
Epoch 157/300
8/8 [==============================] - 0s 4ms/step - loss: 0.0851 - accuracy:
0.9762
Epoch 158/300
8/8 [==============================] - 0s 4ms/step - loss: 0.0540 - accuracy:
0.9881
Epoch 159/300
8/8 [==============================] - 0s 3ms/step - loss: 0.0565 - accuracy:
0.9802
Epoch 160/300
8/8 [==============================] - 0s 3ms/step - loss: 0.0396 - accuracy:
0.9901
Epoch 161/300
8/8 [==============================] - 0s 3ms/step - loss: 0.1107 - accuracy:
0.9623
Epoch 162/300
8/8 [==============================] - 0s 3ms/step - loss: 0.0446 - accuracy:
0.9861
Epoch 163/300
8/8 [==============================] - 0s 3ms/step - loss: 0.0668 - accuracy:
0.9802
Epoch 164/300
8/8 [==============================] - 0s 3ms/step - loss: 0.0365 - accuracy:
0.9960
Epoch 165/300
8/8 [==============================] - 0s 3ms/step - loss: 0.0598 - accuracy:
0.9841
Epoch 166/300
8/8 [==============================] - 0s 2ms/step - loss: 0.0421 - accuracy:
0.9881
Epoch 167/300
8/8 [==============================] - 0s 2ms/step - loss: 0.0697 - accuracy:
0.9861
Epoch 168/300
8/8 [==============================] - 0s 3ms/step - loss: 0.0521 - accuracy:
0.9821
Epoch 169/300
8/8 [==============================] - 0s 2ms/step - loss: 0.0668 - accuracy:
0.9742
Epoch 170/300
8/8 [==============================] - 0s 3ms/step - loss: 0.0560 - accuracy:
0.9802
Epoch 171/300
8/8 [==============================] - 0s 3ms/step - loss: 0.0390 - accuracy:
0.9901
Epoch 172/300
8/8 [==============================] - 0s 3ms/step - loss: 0.0407 - accuracy:
0.9901
```

```
Epoch 173/300
8/8 [==============================] - 0s 2ms/step - loss: 0.0939 - accuracy:
0.9623
Epoch 174/300
8/8 [==============================] - 0s 6ms/step - loss: 0.0628 - accuracy:
0.9782
Epoch 175/300
8/8 [==============================] - 0s 3ms/step - loss: 0.0392 - accuracy:
0.9921
Epoch 176/300
8/8 [==============================] - 0s 4ms/step - loss: 0.0346 - accuracy:
0.9940
Epoch 177/300
8/8 [==============================] - 0s 3ms/step - loss: 0.0842 - accuracy:
0.9702
Epoch 178/300
8/8 [==============================] - 0s 4ms/step - loss: 0.0286 - accuracy:
0.9960
Epoch 179/300
8/8 [==============================] - 0s 2ms/step - loss: 0.0331 - accuracy:
0.9940
Epoch 180/300
8/8 [==============================] - 0s 3ms/step - loss: 0.0340 - accuracy:
0.9940
Epoch 181/300
8/8 [==============================] - 0s 3ms/step - loss: 0.0975 - accuracy:
0.9702
Epoch 182/300
8/8 [==============================] - 0s 2ms/step - loss: 0.0454 - accuracy:
0.9861
Epoch 183/300
8/8 [==============================] - 0s 380us/step - loss: 0.0403 - accuracy:
0.9881
Epoch 184/300
8/8 [==============================] - 0s 2ms/step - loss: 0.0308 - accuracy:
0.9940
Epoch 185/300
8/8 [==============================] - 0s 3ms/step - loss: 0.0370 - accuracy:
0.9921
Epoch 186/300
8/8 [==============================] - 0s 2ms/step - loss: 0.0744 - accuracy:
0.9762
Epoch 187/300
8/8 [==============================] - 0s 3ms/step - loss: 0.0322 - accuracy:
0.9960
Epoch 188/300
8/8 [==============================] - 0s 3ms/step - loss: 0.0445 - accuracy:
0.9881
```

```
Epoch 189/300
8/8 [==============================] - 0s 2ms/step - loss: 0.0360 - accuracy:
0.9901
Epoch 190/300
8/8 [==============================] - 0s 510us/step - loss: 0.0316 - accuracy:
0.9940
Epoch 191/300
8/8 [==============================] - 0s 4ms/step - loss: 0.0807 - accuracy:
0.9702
Epoch 192/300
8/8 [==============================] - 0s 2ms/step - loss: 0.0296 - accuracy:
0.9960
Epoch 193/300
8/8 [==============================] - 0s 4ms/step - loss: 0.0235 - accuracy:
0.9960
Epoch 194/300
8/8 [==============================] - 0s 2ms/step - loss: 0.0379 - accuracy:
0.9940
Epoch 195/300
8/8 [==============================] - 0s 3ms/step - loss: 0.0229 - accuracy:
0.9980
Epoch 196/300
8/8 [==============================] - 0s 2ms/step - loss: 0.0575 - accuracy:
0.9821
Epoch 197/300
8/8 [==============================] - 0s 4ms/step - loss: 0.0235 - accuracy:
0.9980
Epoch 198/300
8/8 [==============================] - 0s 3ms/step - loss: 0.0599 - accuracy:
0.9762
Epoch 199/300
8/8 [==============================] - 0s 5ms/step - loss: 0.0379 - accuracy:
0.9861
Epoch 200/300
8/8 [==============================] - 0s 4ms/step - loss: 0.0318 - accuracy:
0.9921
Epoch 201/300
8/8 [==============================] - 0s 5ms/step - loss: 0.0233 - accuracy:
0.9960
Epoch 202/300
8/8 [==============================] - 0s 5ms/step - loss: 0.0500 - accuracy:
0.9841
Epoch 203/300
8/8 [==============================] - 0s 3ms/step - loss: 0.0205 - accuracy:
1.0000
Epoch 204/300
8/8 [==============================] - 0s 6ms/step - loss: 0.0888 - accuracy:
0.9702
```

```
Epoch 205/300
8/8 [==============================] - 0s 4ms/step - loss: 0.0194 - accuracy:
0.9960
Epoch 206/300
8/8 [==============================] - 0s 2ms/step - loss: 0.0330 - accuracy:
0.9921
Epoch 207/300
8/8 [==============================] - 0s 2ms/step - loss: 0.0282 - accuracy:
0.9921
Epoch 208/300
8/8 [==============================] - 0s 1ms/step - loss: 0.0530 - accuracy:
0.9841
Epoch 209/300
8/8 [==============================] - 0s 4ms/step - loss: 0.0214 - accuracy:
1.0000
Epoch 210/300
8/8 [==============================] - 0s 386us/step - loss: 0.0431 - accuracy:
0.9881
Epoch 211/300
8/8 [==============================] - 0s 2ms/step - loss: 0.0334 - accuracy:
0.9901
Epoch 212/300
8/8 [==============================] - 0s 2ms/step - loss: 0.0241 - accuracy:
0.9960
Epoch 213/300
8/8 [==============================] - 0s 5ms/step - loss: 0.0556 - accuracy:
0.9841
Epoch 214/300
8/8 [==============================] - 0s 3ms/step - loss: 0.0168 - accuracy:
1.0000
Epoch 215/300
8/8 [==============================] - 0s 2ms/step - loss: 0.0320 - accuracy:
0.9921
Epoch 216/300
8/8 [==============================] - 0s 143us/step - loss: 0.0150 - accuracy:
1.0000
Epoch 217/300
8/8 [==============================] - 0s 2ms/step - loss: 0.0218 - accuracy:
1.0000
Epoch 218/300
8/8 [==============================] - 0s 2ms/step - loss: 0.1322 - accuracy:
0.9663
Epoch 219/300
8/8 [==============================] - 0s 3ms/step - loss: 0.0165 - accuracy:
0.9980
Epoch 220/300
8/8 [==============================] - 0s 1ms/step - loss: 0.0143 - accuracy:
1.0000
```

```
Epoch 221/300
8/8 [==============================] - 0s 4ms/step - loss: 0.0185 - accuracy:
0.9980
Epoch 222/300
8/8 [==============================] - 0s 3ms/step - loss: 0.0152 - accuracy:
0.9980
Epoch 223/300
8/8 [==============================] - 0s 2ms/step - loss: 0.0726 - accuracy:
0.9702
Epoch 224/300
8/8 [==============================] - 0s 2ms/step - loss: 0.0158 - accuracy:
1.0000
Epoch 225/300
8/8 [==============================] - 0s 2ms/step - loss: 0.0328 - accuracy:
0.9901
Epoch 226/300
8/8 [==============================] - 0s 2ms/step - loss: 0.0138 - accuracy:
1.0000
Epoch 227/300
8/8 [==============================] - 0s 2ms/step - loss: 0.0138 - accuracy:
0.9980
Epoch 228/300
8/8 [==============================] - 0s 2ms/step - loss: 0.0761 - accuracy:
0.9702
Epoch 229/300
8/8 [==============================] - 0s 2ms/step - loss: 0.0133 - accuracy:
0.9980
Epoch 230/300
8/8 [==============================] - 0s 3ms/step - loss: 0.0127 - accuracy:
1.0000
Epoch 231/300
8/8 [==============================] - 0s 716us/step - loss: 0.1007 - accuracy:
0.9643
Epoch 232/300
8/8 [==============================] - 0s 3ms/step - loss: 0.0136 - accuracy:
0.9980
Epoch 233/300
8/8 [==============================] - 0s 2ms/step - loss: 0.0120 - accuracy:
1.0000
Epoch 234/300
8/8 [==============================] - 0s 96us/step - loss: 0.0125 - accuracy:
1.0000
Epoch 235/300
8/8 [==============================] - 0s 3ms/step - loss: 0.0920 - accuracy:
0.9643
Epoch 236/300
8/8 [==============================] - 0s 3ms/step - loss: 0.0148 - accuracy:
0.9980
```

```
Epoch 237/300
8/8 [==============================] - 0s 3ms/step - loss: 0.0109 - accuracy:
1.0000
Epoch 238/300
8/8 [==============================] - 0s 4ms/step - loss: 0.0118 - accuracy:
1.0000
Epoch 239/300
8/8 [==============================] - 0s 3ms/step - loss: 0.0130 - accuracy:
1.0000
Epoch 240/300
8/8 [==============================] - 0s 2ms/step - loss: 0.0381 - accuracy:
0.9861
Epoch 241/300
8/8 [==============================] - 0s 3ms/step - loss: 0.0252 - accuracy:
0.9921
Epoch 242/300
8/8 [==============================] - 0s 4ms/step - loss: 0.0641 - accuracy:
0.9802
Epoch 243/300
8/8 [==============================] - 0s 2ms/step - loss: 0.0111 - accuracy:
1.0000
Epoch 244/300
8/8 [==============================] - 0s 2ms/step - loss: 0.0728 - accuracy:
0.9722
Epoch 245/300
8/8 [==============================] - 0s 3ms/step - loss: 0.0110 - accuracy:
1.0000
Epoch 246/300
8/8 [==============================] - 0s 3ms/step - loss: 0.0113 - accuracy:
1.0000
Epoch 247/300
8/8 [==============================] - 0s 2ms/step - loss: 0.0111 - accuracy:
1.0000
Epoch 248/300
8/8 [==============================] - 0s 2ms/step - loss: 0.0186 - accuracy:
0.9960
Epoch 249/300
8/8 [==============================] - 0s 2ms/step - loss: 0.0296 - accuracy:
0.9921
Epoch 250/300
8/8 [==============================] - 0s 2ms/step - loss: 0.0121 - accuracy:
1.0000
Epoch 251/300
8/8 [==============================] - 0s 2ms/step - loss: 0.0971 - accuracy:
0.9742
Epoch 252/300
8/8 [==============================] - 0s 2ms/step - loss: 0.0109 - accuracy:
1.0000
```

```
Epoch 253/300
8/8 [==============================] - 0s 2ms/step - loss: 0.0100 - accuracy:
1.0000
Epoch 254/300
8/8 [==============================] - 0s 2ms/step - loss: 0.0092 - accuracy:
1.0000
Epoch 255/300
8/8 [==============================] - 0s 1ms/step - loss: 0.0466 - accuracy:
0.9821
Epoch 256/300
8/8 [==============================] - 0s 3ms/step - loss: 0.0095 - accuracy:
1.0000
Epoch 257/300
8/8 [==============================] - 0s 2ms/step - loss: 0.0081 - accuracy:
1.0000
Epoch 258/300
8/8 [==============================] - 0s 2ms/step - loss: 0.0409 - accuracy:
0.9901
Epoch 259/300
8/8 [==============================] - 0s 3ms/step - loss: 0.0107 - accuracy:
1.0000
Epoch 260/300
8/8 [==============================] - 0s 1ms/step - loss: 0.0084 - accuracy:
1.0000
Epoch 261/300
8/8 [==============================] - 0s 3ms/step - loss: 0.0106 - accuracy:
1.0000
Epoch 262/300
8/8 [==============================] - 0s 1ms/step - loss: 0.0798 - accuracy:
0.9762
Epoch 263/300
8/8 [==============================] - 0s 2ms/step - loss: 0.0077 - accuracy:
1.0000
Epoch 264/300
8/8 [==============================] - 0s 2ms/step - loss: 0.0099 - accuracy:
1.0000
Epoch 265/300
8/8 [==============================] - 0s 4ms/step - loss: 0.0094 - accuracy:
1.0000
Epoch 266/300
8/8 [==============================] - 0s 5ms/step - loss: 0.0547 - accuracy:
0.9841
Epoch 267/300
8/8 [==============================] - 0s 3ms/step - loss: 0.0252 - accuracy:
0.9921
Epoch 268/300
8/8 [==============================] - 0s 1ms/step - loss: 0.0086 - accuracy:
1.0000
```

```
Epoch 269/300
8/8 [==============================] - 0s 3ms/step - loss: 0.0334 - accuracy:
0.9901
Epoch 270/300
8/8 [==============================] - 0s 6ms/step - loss: 0.0083 - accuracy:
1.0000
Epoch 271/300
8/8 [==============================] - 0s 4ms/step - loss: 0.0076 - accuracy:
1.0000
Epoch 272/300
8/8 [==============================] - 0s 2ms/step - loss: 0.0223 - accuracy:
0.9940
Epoch 273/300
8/8 [==============================] - 0s 1ms/step - loss: 0.0950 - accuracy:
0.9742
Epoch 274/300
8/8 [==============================] - 0s 4ms/step - loss: 0.0067 - accuracy:
1.0000
Epoch 275/300
8/8 [==============================] - 0s 3ms/step - loss: 0.0070 - accuracy:
1.0000
Epoch 276/300
8/8 [==============================] - 0s 3ms/step - loss: 0.0066 - accuracy:
1.0000
Epoch 277/300
8/8 [==============================] - 0s 2ms/step - loss: 0.0082 - accuracy:
1.0000
Epoch 278/300
8/8 [==============================] - 0s 127us/step - loss: 0.0543 - accuracy:
0.9762
Epoch 279/300
8/8 [==============================] - 0s 2ms/step - loss: 0.0074 - accuracy:
1.0000
Epoch 280/300
8/8 [==============================] - 0s 2ms/step - loss: 0.0069 - accuracy:
1.0000
Epoch 281/300
8/8 [==============================] - 0s 4ms/step - loss: 0.0065 - accuracy:
1.0000
Epoch 282/300
8/8 [==============================] - 0s 5ms/step - loss: 0.0733 - accuracy:
0.9742
Epoch 283/300
8/8 [==============================] - 0s 4ms/step - loss: 0.0060 - accuracy:
1.0000
Epoch 284/300
8/8 [==============================] - 0s 4ms/step - loss: 0.0061 - accuracy:
1.0000
```

```
Epoch 285/300
8/8 [==============================] - 0s 2ms/step - loss: 0.0884 - accuracy:
0.9782
Epoch 286/300
8/8 [==============================] - 0s 3ms/step - loss: 0.0103 - accuracy:
1.0000
Epoch 287/300
8/8 [==============================] - 0s 4ms/step - loss: 0.0059 - accuracy:
1.0000
Epoch 288/300
8/8 [==============================] - 0s 2ms/step - loss: 0.0059 - accuracy:
1.0000
Epoch 289/300
8/8 [==============================] - 0s 2ms/step - loss: 0.0057 - accuracy:
1.0000
Epoch 290/300
8/8 [==============================] - 0s 4ms/step - loss: 0.0544 - accuracy:
0.9841
Epoch 291/300
8/8 [==============================] - 0s 2ms/step - loss: 0.0056 - accuracy:
1.0000
Epoch 292/300
8/8 [==============================] - 0s 3ms/step - loss: 0.0054 - accuracy:
1.0000
Epoch 293/300
8/8 [==============================] - 0s 2ms/step - loss: 0.0821 - accuracy:
0.9782
Epoch 294/300
8/8 [==============================] - 0s 3ms/step - loss: 0.0055 - accuracy:
1.0000
Epoch 295/300
8/8 [==============================] - 0s 5ms/step - loss: 0.0058 - accuracy:
1.0000
Epoch 296/300
8/8 [==============================] - 0s 5ms/step - loss: 0.0051 - accuracy:
1.0000
Epoch 297/300
8/8 [==============================] - 0s 2ms/step - loss: 0.0331 - accuracy:
0.9881
Epoch 298/300
8/8 [==============================] - 0s 2ms/step - loss: 0.0579 - accuracy:
0.9821
Epoch 299/300
8/8 [==============================] - 0s 3ms/step - loss: 0.0051 - accuracy:
1.0000
Epoch 300/300
8/8 [==============================] - 0s 1ms/step - loss: 0.0049 - accuracy:
1.0000
```

```
16/16 [==============================] - 0s 1ms/step
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        76
           1       1.00      1.00      1.00        71
           2       1.00      1.00      1.00        71
           3       1.00      1.00      1.00        76
           4       1.00      1.00      1.00        67
           5       1.00      1.00      1.00        67
           6       1.00      1.00      1.00        76

    accuracy                           1.00       504
   macro avg       1.00      1.00      1.00       504
weighted avg       1.00      1.00      1.00       504
```

4. Contesta lo siguientes:

A. ¿Observas alguna mejora importante al optimizar el tamaño de la red? ¿Es el resultado que esperabas?

En efecto, hay una mejora significativa después de optimizar el tamaño de la red neuronal. Con 4 capas de 30 neuronas se obtuvo una presición de 0.9619. Al evaluar con la mejor configuración [50, 20] tenemos una presición del 0.966666.

B. ¿Qué inconvenientes hay al encontrar el tamaño óptimo de la red? ¿Por qué?

Mayor tiempo computacional, mayor probabilidad de sobreajuste y que cambian los valores de los hiperparámetros

0.0.4 EJERCICIO 3

Se tienen datos procesados de un experimento de psicología en el que se mide la respuesta cerebral cuando un sujeto presta atención a un estímulo visual que aparece de manera repentina y cuando no presta atención a dicho estímulo visual. Los datos están en archivos de texto, los cuales se cargan con la función loadtxt de numpy (https://numpy.org/doc/stable/reference/generated/numpy.loadtxt.html).

La primera columna corresponde a la clase (1 o 2). La clase 1 representa cuando el sujeto está prestando atención, y la clase 2 cuando no lo hace. La segunda columna se ignora, mientras que el resto de las columnas indican las variables que se calcularon de la respuesta cerebral medida con la técnicas de Electroencefaolografía para cada caso.

Para tu conjunto de datos:

1. Implementa un modelo perceptrón de una neurona entrenado con descenso de gradiente estocástico, y evalúalo con validación cruzada. Para esta caso, es necesario que encuentres la gráfica de Época Vs Exactitud.

2. Repite el paso anterior, pero utilizando descenso de gradiente de lote y de mini-lote para entrenar el modelo.

3. Evalúa un modelo perceptrón multicapa con validación cruzada. Para este caso, puedes utilizar un modelo dado por scikit-learn, Keras o Pytorch.

4. ¿El modelo de una neurona es suficiente para modelar el conjunto de datos de este problema?

[52]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import StratifiedKFold, cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers.legacy import SGD
```

[53]:
```
df3 = pd.read_csv('P1_5.txt', sep='\t', header=None)
df3.shape
```

[53]:
```
(1970, 156)
```

[54]:
```
df3.head(5)
```

[54]:

	0	1	2	3	4	5	6	7	\
0	1	0	6.778645	5.554810	4.935933	5.639114	6.603463	6.309278	
1	1	1	-0.354725	-0.210143	-0.337065	-0.613978	-0.725109	-0.509282	
2	1	1	1.727716	2.056163	1.876898	0.992565	0.224456	0.429907	
3	1	1	1.064721	1.299112	0.705400	0.422355	1.269587	2.672225	
4	1	1	-0.010284	-0.097968	-0.345274	-0.216691	0.351261	0.781839	

	8	9	…	146	147	148	149	150	\
0	4.738829	3.110043	…	-0.609716	-0.822197	-0.109120	0.899585	1.374105	
1	-0.154146	0.062651	…	-0.052648	0.544180	0.781654	0.833143	1.253793	
2	1.229387	1.403841	…	-0.827163	0.208835	1.473647	1.545710	0.672175	
3	3.240471	2.234018	…	-0.726883	-0.768780	-0.600054	-0.948638	-1.688718	
4	0.695825	0.298840	…	0.730151	-0.396311	-0.324040	0.629645	1.199808	

	151	152	153	154	155
0	1.187913	0.742605	0.347446	0.131211	NaN
1	1.914558	2.013101	1.287430	0.521264	NaN
2	0.097850	0.288585	0.679454	0.822155	NaN
3	-1.858080	-0.862087	0.660512	1.426284	NaN
4	0.600917	-0.806892	-2.087005	-2.579933	NaN

[5 rows x 156 columns]

[56]:
```
# Separar la variable objetivo y las características
y = df3.iloc[:, 0]
X = df3.iloc[:, 2:-1]
```

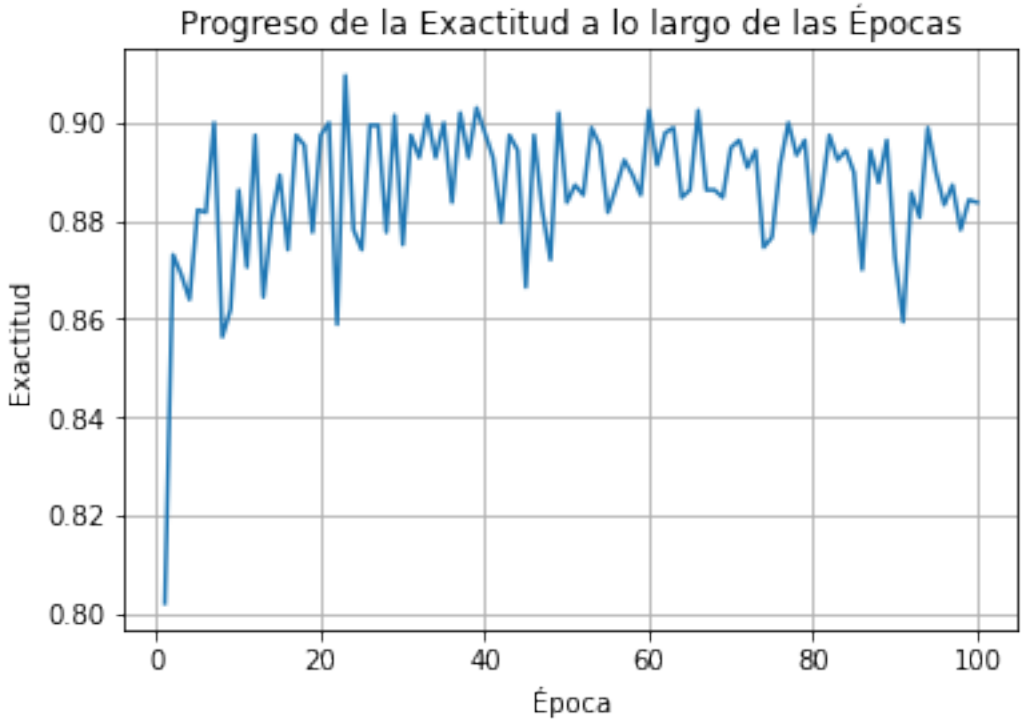
```
# Aplanar las características si es necesario
if len(caracteristicas.shape) == 3:
    n_samples, timesteps, n_features = caracteristicas.shape
    caracteristicas = caracteristicas.reshape(n_samples, timesteps * n_features)

# Contar las instancias para cada etiqueta
cantidad_etiqueta_1 = objetivo.value_counts().get(1, 0)
cantidad_etiqueta_2 = objetivo.value_counts().get(2, 0)

print("Cantidad de datos con etiqueta 1:", cantidad_etiqueta_1)
print("Cantidad de datos con etiqueta 2:", cantidad_etiqueta_2)
```

```
Cantidad de datos con etiqueta 1: 281
Cantidad de datos con etiqueta 2: 1689
```

1. Implementa un modelo perceptrón de una neurona entrenado con descenso de gradiente estocástico, y evalúalo con validación cruzada. Para esta caso, es necesario que encuentres la gráfica de Época Vs Exactitud.

[57]:
```
from sklearn.linear_model import SGDClassifier
from sklearn.model_selection import StratifiedKFold

# Definir el modelo de Perceptrón con SGD (una sola neurona)
modelo_sgd = SGDClassifier(loss='perceptron', max_iter=1, tol=None,␣
 ↪random_state=42, learning_rate='constant', eta0=0.01)

# Crear una tubería que incluya la normalización de los datos
modelo_completo = make_pipeline(StandardScaler(), modelo_sgd)

# Especificar el número de épocas y preparar para almacenar las exactitudes
epocas = 100
exactitudes = []

# Configurar la validación cruzada estratificada
validacion_cruzada = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

# Entrenar y evaluar el modelo en cada época
for epoca in range(1, epocas + 1):
    modelo_completo.named_steps['sgdclassifier'].max_iter = epoca
    puntajes = cross_val_score(modelo_completo, caracteristicas, objetivo,␣
 ↪cv=validacion_cruzada, scoring='accuracy', n_jobs=-1)
    exactitudes.append(puntajes.mean())

# Graficar el progreso de la exactitud a lo largo de las épocas
plt.plot(range(1, epocas + 1), exactitudes)
plt.title('Progreso de la Exactitud a lo largo de las Épocas')
plt.xlabel('Época')
```

```
plt.ylabel('Exactitud')
plt.grid(True)
plt.show()
```



2. Repite el paso anterior, pero utilizando descenso de gradiente de lote y de mini-lote para entrenar el modelo.

[58]:
```
from sklearn.metrics import accuracy_score

# Normalizar los datos
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Configuración para la validación cruzada
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

# Función para crear el modelo de perceptrón
def create_perceptron_model(learning_rate, batch_size):
    model = Sequential()
    model.add(Dense(1, input_dim=X_scaled.shape[1], activation='relu'))
    model.compile(optimizer=SGD(learning_rate=learning_rate),␣
 ↪loss='binary_crossentropy', metrics=['accuracy'])
    return model
```

```python
# Evaluar el modelo con validación cruzada
def evaluate_model(X, y, learning_rate, batch_size):
    accuracies = []

    for train_index, test_index in skf.split(X,y):
        X_train, X_test = X[train_index], X[test_index]
        y_train, y_test = y[train_index], y[test_index]

        # Crear y entrenar el modelo
        model = create_perceptron_model(learning_rate, batch_size)
        model.fit(X_train, y_train, epochs=50, batch_size=batch_size, verbose=0)

        # Evaluar el modelo
        y_pred = (model.predict(X_test) > 0.5).astype("int32")
        accuracy = accuracy_score(y_test, y_pred)
        accuracies.append(accuracy)

    return np.mean(accuracies)

# Evaluar con descenso de gradiente en lote
batch_size = X_scaled.shape[0]
print("Evaluación con descenso de gradiente en lote:")
accuracy_batch = evaluate_model(X_scaled, y, learning_rate=0.001,␣
 ↪batch_size=batch_size)
print(f"Exactitud media: {accuracy_batch:.4f}")

# Evaluar con descenso de gradiente en mini-lote
print("\nEvaluación con descenso de gradiente en mini-lote:")
accuracy_mini_batch = evaluate_model(X_scaled, y, learning_rate=0.001,␣
 ↪batch_size=10)
print(f"Exactitud media: {accuracy_mini_batch:.4f}")
```

```
Evaluación con descenso de gradiente en lote:
13/13 [==============================] - 0s 1ms/step
13/13 [==============================] - 0s 537us/step
13/13 [==============================] - 0s 2ms/step
13/13 [==============================] - 0s 1ms/step
13/13 [==============================] - 0s 0s/step
Exactitud media: 0.0604

Evaluación con descenso de gradiente en mini-lote:
13/13 [==============================] - 0s 814us/step
13/13 [==============================] - 0s 1ms/step
13/13 [==============================] - 0s 1ms/step
13/13 [==============================] - 0s 1ms/step
13/13 [==============================] - 0s 0s/step
```

```
Exactitud media: 0.0959
```

3. Evalúa un modelo perceptrón multicapa con validación cruzada. Para este caso, puedes utilizar un modelo dado por scikit-learn, Keras o Pytorch.

[59]:
```
# Normalizar los datos
scaler = StandardScaler()
X_scaled = scaler.fit_transform(caracteristicas)

# Configuración para la validación cruzada
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

# Crear el modelo MLP
model = MLPClassifier(hidden_layer_sizes=(6,6,6), activation='relu',␣
 ↪solver='adam', max_iter=1000, random_state=42)

# Evaluar el modelo con validación cruzada
scores = cross_val_score(model, X_scaled, objetivo, cv=skf, scoring='accuracy')

# Resultados
print("Exactitud media con validación cruzada: {:.4f}".format(np.mean(scores)))
```

```
Exactitud media con validación cruzada: 0.9020
```

4. ¿El modelo de una neurona es suficiente para modelar el conjunto de datos de este problema?

Relativamente si es suficiente, sin embargo es mucho mejor mientras más capas tenga ya que incrementa la exactitud media conforme se va haciendo cada vez más complejo.