Step1: Import libraries

```
# Import libraries

import numpy as np
import pandas as pd
import matplotlib.cm as cm
import matplotlib.pyplot as plt

import plotly.express as px
import plotly.graph_objs as go

import tensorflow as tf

from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split

#import warnings
#warnings.filterwarnings('ignore')
```

```
pip install -U kaleido
```

```
Requirement already satisfied: kaleido in /usr/local/lib/python3.10/dist-packages (0.2.1)
```

Step2: Load "Fertility_Diagnosis.txt" file.

```
fertility_data = np.genfromtxt('Fertility_Diagnosis.txt', delimiter=',')
print(f'Shape = {fertility_data.shape} \n')
pd.DataFrame(fertility_data).head()
```

```
Shape = (100, 10)
```

	0	1	2	3	4	5	6	7	8	9
0	-0.33	0.69	0.0	1.0	1.0	0.0	0.8	0.0	0.88	0.0
1	-0.33	0.94	1.0	0.0	1.0	0.0	0.8	1.0	0.31	1.0
2	-0.33	0.50	1.0	0.0	0.0	0.0	1.0	-1.0	0.50	0.0
3	-0.33	0.75	0.0	1.0	1.0	0.0	1.0	-1.0	0.38	0.0
4	-0.33	0.67	1.0	1.0	0.0	0.0	0.8	-1.0	0.50	1.0

Create training and testing datasets.

```
input_features = fertility_data[:, :-1]
true_output = fertility_data[:, -1]
```

```
X_train, X_test, y_train, y_test = train_test_split(input_features, true_output, test_size = 0.2, random_state = 42)
```

```
print(f'Shape = {X_train.shape} \n')
pd.DataFrame(X_train).head()
```

```
Shape = (80, 9)
```

	0	1	2	3	4	5	6	7	8
0	-0.33	0.58	1.0	0.0	1.0	0.0	0.8	1.0	0.19
1	-0.33	0.83	1.0	1.0	1.0	0.0	1.0	-1.0	0.31
2	1.00	0.67	1.0	0.0	1.0	0.0	0.6	-1.0	0.38
3	-1.00	0.58	1.0	0.0	1.0	-1.0	0.8	1.0	0.50
4	-0.33	0.50	1.0	0.0	1.0	-1.0	0.8	-1.0	0.50

```
print(f'Shape = {y_train.shape} \n')
pd.Series(y_train, name = 'Output').head()
```

```
Shape = (80,)

0    0.0
1    0.0
2    1.0
3    0.0
4    0.0
Name: Output, dtype: float64
```

```
y_train = y_train.reshape(y_train.shape[0], 1)
print(f'Shape = {y_train.shape} \n')
```

```
Shape = (80, 1)
```

```
print(f'Shape = {X_test.shape} \n')
pd.DataFrame(X_test).head()
```

```
Shape = (20, 9)
```

	0	1	2	3	4	5	6	7	8
0	-0.33	0.86	1.0	1.0	1.0	1.0	1.0	-1.0	0.25
1	-0.33	0.58	1.0	1.0	1.0	-1.0	0.8	0.0	0.19
2	-0.33	0.50	1.0	1.0	0.0	-1.0	0.8	0.0	0.88
3	-1.00	0.53	1.0	0.0	0.0	1.0	1.0	0.0	0.44

```
print(f'Shape = {y_test.shape} \n')
pd.Series(y_test, name = 'Output').head()
```

```
Shape = (20,)

0    0.0
1    0.0
2    1.0
3    0.0
4    0.0
Name: Output, dtype: float64
```

```
y_test = y_test.reshape(y_test.shape[0], 1)
print(f'Shape = {y_test.shape} \n')
```

```
Shape = (20, 1)
```

Step3: Define input layer size and output layer size. Set error tolerance value.

```
input_size = X_train.shape[1]
output_size = 1
```

```
ERROR_TOLERANCE = 0.05
```

```
LEARNING_RATE = 0.001
```

```
# Lists to store training error
training_error_list = []

# Lists to store testing error
testing_error_list = []
```

Step4: Define feedforward network for training data.

```
def sigmoid(x):
    return 1 / (1 + np.exp(-x))
```

```
def feedforward(input_data, weights_first_layer, weights_second_layer):
  output_first_layer = tf.sigmoid(tf.matmul(tf.transpose(input_data.reshape(-1, 1)), weights_first_layer))     # Can use np.dot or tf.matmul
  output_second_layer = tf.sigmoid(tf.matmul(output_first_layer, weights_second_layer))
  return output_first_layer, output_second_layer
```

```
def feedforward_np(input_data, weights_first_layer, weights_second_layer):
  output_first_layer = sigmoid(np.dot(input_data.T, weights_first_layer))     # Can use np.dot or tf.matmul
  output_second_layer = sigmoid(np.dot(output_first_layer, weights_second_layer))
  return output_first_layer, output_second_layer
```

Step5: Perform backpropagation.

```
def cost_function(true_output, layer2_output):
  return true_output.reshape(-1, 1) - layer2_output
```

```
def sigmoid_derivative(x):
  return x * (1 - x)
```

```
def backpropogate(input, true_output, layer1_output, layer2_output, weights_layer1, weights_layer2):
  layer_2_error = cost_function(true_output, layer2_output)
  layer_2_delta = layer_2_error * sigmoid_derivative(layer2_output)
  weights_layer2 += LEARNING_RATE * tf.matmul(tf.reshape(layer1_output, shape = [layer1_output.shape[0], 1]), layer_2_delta)

  layer_1_error = tf.matmul(layer_2_delta, tf.transpose(weights_layer2))
  layer_1_delta = layer_1_error * sigmoid_derivative(layer1_output)
  weights_layer1 += LEARNING_RATE * tf.matmul(input.reshape(-1, 1), layer_1_delta)

  return weights_layer1, weights_layer2
```

```
def backpropogate_np(input, true_output, layer1_output, layer2_output, weights_layer1, weights_layer2):
  layer_2_error = cost_function(true_output, layer2_output)
  layer_2_delta = layer_2_error * sigmoid_derivative(layer2_output)
  weights_layer2 += LEARNING_RATE * np.dot(layer1_output.reshape(layer1_output.shape[0], 1), layer_2_delta)

  layer_1_error = np.dot(layer_2_delta, weights_layer2.T)
  layer_1_delta = layer_1_error * sigmoid_derivative(layer1_output)
  weights_layer1 += LEARNING_RATE * np.dot(input, layer_1_delta)

  return weights_layer1, weights_layer2
```

Step6: Check the convergence and print the debug data for number of epochs ranging from 1 to 1000001.

```
MAX_EPOCH = 1000001
```

```
def fit_mlp(X_train, y_train, weights_layer1, weights_layer2):
  current_error = 0
  flag = 0

  alive_cnt = 0

  for epoch in range(1, MAX_EPOCH + 1):
    all_layer2_output = []
    for idx, input in enumerate(X_train):

      input = input.reshape(-1, 1)

      # Feedforward
      #layer1_output, layer2_output = feedforward(input, weights_layer1, weights_layer2)
      layer1_output, layer2_output = feedforward_np(input, weights_layer1, weights_layer2)


      # Backpropagation
      true_output = y_train[idx]
      #weights_layer1, weights_layer2 = backpropogate(input, true_output, layer1_output, layer2_output, weights_layer1, weights_layer2)
      weights_layer1, weights_layer2 = backpropogate_np(input, true_output, layer1_output, layer2_output, weights_layer1, weights_layer2)


      all_layer2_output.append(layer2_output)

    # Progress update
    '''if epoch % 1000 == 0:
      print(f'Alive {alive_cnt} - Epoch:{epoch}')
      alive_cnt += 1'''

    # Check convergence
    if epoch % 100000 == 0:
        current_mse = mean_squared_error(y_train, np.array(all_layer2_output).reshape(-1, 1))
        print(f'Epoch {epoch}: MSE = {current_mse}')
        if current_mse < ERROR_TOLERANCE:
          flag = 1
          print(f'Model converged at Epoch {epoch} with MSE = {current_mse}')
          break

  if flag == 0:
    print('Model did not converge.')

  return current_error
```

```
def fit_mlp_np(X_train, y_train, weights_layer1, weights_layer2):
  current_error = 0
  flag = 0

  alive_cnt = 0

  for epoch in range(1, MAX_EPOCH + 1):
    #all_layer2_output = []
    input = input.reshape(-1, 1)

    # Feedforward
    #layer1_output, layer2_output = feedforward(input, weights_layer1, weights_layer2)
    layer1_output, layer2_output = feedforward_np(input, weights_layer1, weights_layer2)


    # Backpropagation
    #true_output = y_train[idx]
    #weights_layer1, weights_layer2 = backpropogate(input, true_output, layer1_output, layer2_output, weights_layer1, weights_layer2)
    weights_layer1, weights_layer2 = backpropogate_np(input, y_train, layer1_output, layer2_output, weights_layer1, weights_layer2)


    #all_layer2_output.append(layer2_output)

    # Progress update
    '''if epoch % 1000 == 0:
      print(f'Alive {alive_cnt} - Epoch:{epoch}')
      alive_cnt += 1'''

    # Check convergence
    if epoch % 100000 == 0:
        current_mse = mean_squared_error(y_train, layer2_output).reshape(-1, 1)
        print(f'Epoch {epoch}: MSE = {current_mse}')
        if current_mse < ERROR_TOLERANCE:
          flag = 1
          print(f'Model converged at Epoch {epoch} with MSE = {current_mse}')
          break

  if flag == 0:
    print('Model did not converge.')

  return current_error
```

Step7: Evaluate the training results.

```
training_errors = []

for hidden_layer_size in range(1, 10):
    print(f'Training with {hidden_layer_size} neuron(s) in the hidden layer\n')

    np.random.seed(0)
    initial_weights_layer1 = 2 * np.random.random((input_size, hidden_layer_size)) - 1
    initial_weights_layer2 = 2 * np.random.random((hidden_layer_size, output_size)) - 1

    training_error = fit_mlp(X_train, y_train, initial_weights_layer1, initial_weights_layer2)
    print(f'Training Mean Squared Error (hidden_layer_size = {hidden_layer_size}): {training_error}')
    training_errors.append(training_error)
```

```
Training with 1 neuron(s) in the hidden layer

Alive 0 - Epoch:1000
Alive 1 - Epoch:2000
Alive 2 - Epoch:3000
Alive 3 - Epoch:4000
Alive 4 - Epoch:5000
Alive 5 - Epoch:6000
Alive 6 - Epoch:7000
```



```
training_errors = []

for hidden_layer_size in range(1, 10):
    print(f'Training with {hidden_layer_size} neuron(s) in the hidden layer\n')

    np.random.seed(0)
    initial_weights_layer1 = 2 * np.random.random((input_size, hidden_layer_size)) - 1
    initial_weights_layer2 = 2 * np.random.random((hidden_layer_size, output_size)) - 1

    training_error = fit_mlp(X_train, y_train, initial_weights_layer1, initial_weights_layer2)
    print(f'Training Mean Squared Error (hidden_layer_size = {hidden_layer_size}): {training_error}')
    training_errors.append(training_error)
```



```
Training with 1 neuron(s) in the hidden layer

Alive 0 - Epoch:1000
Alive 1 - Epoch:2000
Alive 2 - Epoch:3000
Alive 3 - Epoch:4000
Alive 4 - Epoch:5000
Alive 5 - Epoch:6000
Alive 6 - Epoch:7000
Alive 7 - Epoch:8000
Alive 8 - Epoch:9000
Alive 9 - Epoch:10000
Alive 10 - Epoch:11000
Alive 11 - Epoch:12000
Alive 12 - Epoch:13000
Alive 13 - Epoch:14000
Alive 14 - Epoch:15000
Alive 15 - Epoch:16000
Alive 16 - Epoch:17000
Alive 17 - Epoch:18000
Alive 18 - Epoch:19000
Alive 19 - Epoch:20000
Alive 20 - Epoch:21000
Alive 21 - Epoch:22000
Alive 22 - Epoch:23000
Alive 23 - Epoch:24000
Alive 24 - Epoch:25000
Alive 25 - Epoch:26000
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-22-297b5a150252> in <cell line: 3>()
      8     initial_weights_layer2 = 2 * np.random.random((hidden_layer_size, output_size)) - 1
      9 
---> 10     training_error = fit_mlp(X_train, y_train, initial_weights_layer1, initial_weights_layer2)
     11     print(f'Training Mean Squared Error (hidden_layer_size = {hidden_layer_size}): {training_error}')
     12     training_errors.append(training_error)
```

2 frames

```
/usr/local/lib/python3.10/dist-packages/tensorflow/python/ops/weak_tensor_ops.py in wrapper(*args, **kwargs)
     85 
     86   def wrapper(*args, **kwargs):
---> 87     if not ops.is_auto_dtype_conversion_enabled():
     88       return op(*args, **kwargs)
     89     bound_arguments = signature.bind(*args, **kwargs)

KeyboardInterrupt: 
```

SEARCH STACK OVERFLOW


```
training_errors = []

for hidden_layer_size in range(1, 10):
    print(f'Training with {hidden_layer_size} neuron(s) in the hidden layer\n')

    np.random.seed(0)
    initial_weights_layer1 = 2 * np.random.random((input_size, hidden_layer_size)) - 1
    initial_weights_layer2 = 2 * np.random.random((hidden_layer_size, output_size)) - 1

    training_error = fit_mlp(X_train, y_train, initial_weights_layer1, initial_weights_layer2)
    print(f'Training Mean Squared Error (hidden_layer_size = {hidden_layer_size}): {training_error}')
    training_errors.append(training_error)
```

```
Training with 1 neuron(s) in the hidden layer

Epoch 100000: MSE = 0.0789841077883142
Epoch 200000: MSE = 0.07611599902984909
Epoch 300000: MSE = 0.07443080138392397
Epoch 400000: MSE = 0.0732560827412726
```

```
# Sequential Code
import time

training_errors = []
testing_errors = []

start_time = time.perf_counter()

for hidden_layer_size in range(1, 10):
    print(f'Training with {hidden_layer_size} neuron(s) in the hidden layer\n')

    np.random.seed(0)
    weights_layer1 = 2 * np.random.random((input_size, hidden_layer_size)) - 1
    weights_layer2 = 2 * np.random.random((hidden_layer_size, output_size)) - 1


    current_error = 0
    flag = 0

    alive_cnt = 0

    for epoch in range(1, MAX_EPOCH + 1):
      input = X_train

      # Feedforward
      layer1_output = sigmoid(np.dot(input, weights_layer1))     # Can use np.dot or tf.matmul
      layer2_output = sigmoid(np.dot(layer1_output, weights_layer2))


      # Backpropagation
      layer_2_error = cost_function(y_train, layer2_output)
      layer_2_delta = layer_2_error * sigmoid_derivative(layer2_output)
      weights_layer2 += LEARNING_RATE * np.dot(layer1_output.T, layer_2_delta)

      layer_1_error = np.dot(layer_2_delta, weights_layer2.T)
      layer_1_delta = layer_1_error * sigmoid_derivative(layer1_output)
      weights_layer1 += LEARNING_RATE * np.dot(input.T, layer_1_delta)


      # Check convergence
      if epoch % 100000 == 0:
          current_mse = mean_squared_error(y_train, layer2_output).reshape(-1, 1)
          print(f'Epoch {epoch}: MSE = {current_mse}')
          if current_mse < ERROR_TOLERANCE:
            flag = 1
            print(f'Model converged at Epoch {epoch} with MSE = {current_mse}\n')
            break

    if flag == 0:
      print('Model did not converge.\n')


    training_input = X_train
    training_layer_1_output = sigmoid(np.dot(training_input, weights_layer1))
    training_layer_2_output = sigmoid(np.dot(training_layer_1_output, weights_layer2))

    training_error = mean_squared_error(y_train, training_layer_2_output).reshape(-1, 1)
    print(f'Training Mean Squared Error (hidden_layer_size = {hidden_layer_size}): {current_error}')
    training_errors.append(training_error)



    testing_input = X_test
    testing_layer_1_output = sigmoid(np.dot(testing_input, weights_layer1))
    testing_layer_2_output = sigmoid(np.dot(testing_layer_1_output, weights_layer2))

    testing_error = mean_squared_error(y_test, testing_layer_2_output).reshape(-1, 1)
    print(f'Testing Mean Squared Error (hidden_layer_size = {hidden_layer_size}): {current_error}')
    print('-----------------------------------------------------------------------------------------')
    testing_errors.append(testing_error)

end_time = time.perf_counter()

print(f'Time taken = {end_time - start_time} seconds')
```

```
Testing Mean Squared Error (hidden_layer_size = 5): 0
-----------------------------------------------------------------------------------------
Training with 6 neuron(s) in the hidden layer

Epoch 100000: MSE = [[0.0288676]]
Model converged at Epoch 100000 with MSE = [[0.0288676]]

Training Mean Squared Error (hidden_layer_size = 6): 0
Testing Mean Squared Error (hidden_layer_size = 6): 0
-----------------------------------------------------------------------------------------
Training with 7 neuron(s) in the hidden layer

Epoch 100000: MSE = [[0.03568713]]
Model converged at Epoch 100000 with MSE = [[0.03568713]]

Training Mean Squared Error (hidden_layer_size = 7): 0
Testing Mean Squared Error (hidden_layer_size = 7): 0
-----------------------------------------------------------------------------------------
Training with 8 neuron(s) in the hidden layer

Epoch 100000: MSE = [[0.02682119]]
Model converged at Epoch 100000 with MSE = [[0.02682119]]

Training Mean Squared Error (hidden_layer_size = 8): 0
Testing Mean Squared Error (hidden_layer_size = 8): 0
-----------------------------------------------------------------------------------------
Training with 9 neuron(s) in the hidden layer

Epoch 100000: MSE = [[0.02891496]]
Model converged at Epoch 100000 with MSE = [[0.02891496]]

Training Mean Squared Error (hidden_layer_size = 9): 0
Testing Mean Squared Error (hidden_layer_size = 9): 0
-----------------------------------------------------------------------------------------
Time taken = 94.37689347699961 seconds
```

```
training_errors
```

```
[array([[0.0691306]]),
 array([[0.04086159]]),
 array([[0.03319421]]),
 array([[0.03432255]]),
 array([[0.03949911]]),
 array([[0.02886735]]),
 array([[0.03568698]]),
 array([[0.02682096]]),
 array([[0.02891473]])]
```

```
len(training_errors)
```

```
9
```

```
training_errors = np.array(list(map(np.ravel, training_errors[:])))
training_errors
```

```
array([[0.0691306 ],
       [0.04086159],
       [0.03319421],
       [0.03432255],
       [0.03949911],
       [0.02886735],
       [0.03568698],
       [0.02682096],
       [0.02891473]])
```

```
training_errors = training_errors.flatten()
training_errors
```

```
array([0.0691306 , 0.04086159, 0.03319421, 0.03432255, 0.03949911,
       0.02886735, 0.03568698, 0.02682096, 0.02891473])
```

Step8: Evaluate testing results.

```
testing_errors = []

for hidden_layer_size in range(1, 10):

    np.random.seed(0)
    initial_weights_layer1 = 2 * np.random.random((input_size, hidden_layer_size)) - 1
    initial_weights_layer2 = 2 * np.random.random((hidden_layer_size, output_size)) - 1

    testing_error = fit_mlp(X_test, y_test, initial_weights_layer1, initial_weights_layer2)
    print(f'Testing Mean Squared Error (hidden_layer_size = {hidden_layer_size}): {testing_error}')
    testing_errors.append(testing_error)
```

```
testing_errors
```

```
[array([[0.1018304]]),
 array([[0.10114237]]),
 array([[0.10127125]]),
 array([[0.10221508]]),
 array([[0.10479307]]),
 array([[0.10397935]]),
 array([[0.10205914]]),
 array([[0.10162091]]),
 array([[0.10144787]])]
```

```
testing_errors = np.array(list(map(np.ravel, testing_errors[:])))
testing_errors
```

```
array([[0.1018304 ],
       [0.10114237],
       [0.10127125],
       [0.10221508],
       [0.10479307],
       [0.10397935],
       [0.10205914],
       [0.10162091],
       [0.10144787]])
```

```
testing_errors = testing_errors.flatten()
testing_errors
```

```
array([0.1018304 , 0.10114237, 0.10127125, 0.10221508, 0.10479307,
       0.10397935, 0.10205914, 0.10162091, 0.10144787])
```

Start coding or generate with AI.

Step9: Plot the results.

```
x_axis_str = 'Number of Neurons in the Hidden layer'
y_axis_str = 'Mean Squared Error (MSE)'

fig = px.line(x = range(1,10), y = [training_errors, testing_errors],
              width = 750,
              height = 500,
              title = 'Neural Network Performance with a Single Hidden Layer for Fertility Classification',
              labels = {'x' : x_axis_str})

fig.update_traces(name = 'Train Error', selector = dict(name = 'wide_variable_0'))
fig.update_traces(name = 'Test Error', selector = dict(name = 'wide_variable_1'))
fig.update_layout(yaxis_title = y_axis_str)
fig.show()
fig.write_image('NN Performance with a Single Hidden Layer for Fertility Classification.png')
```