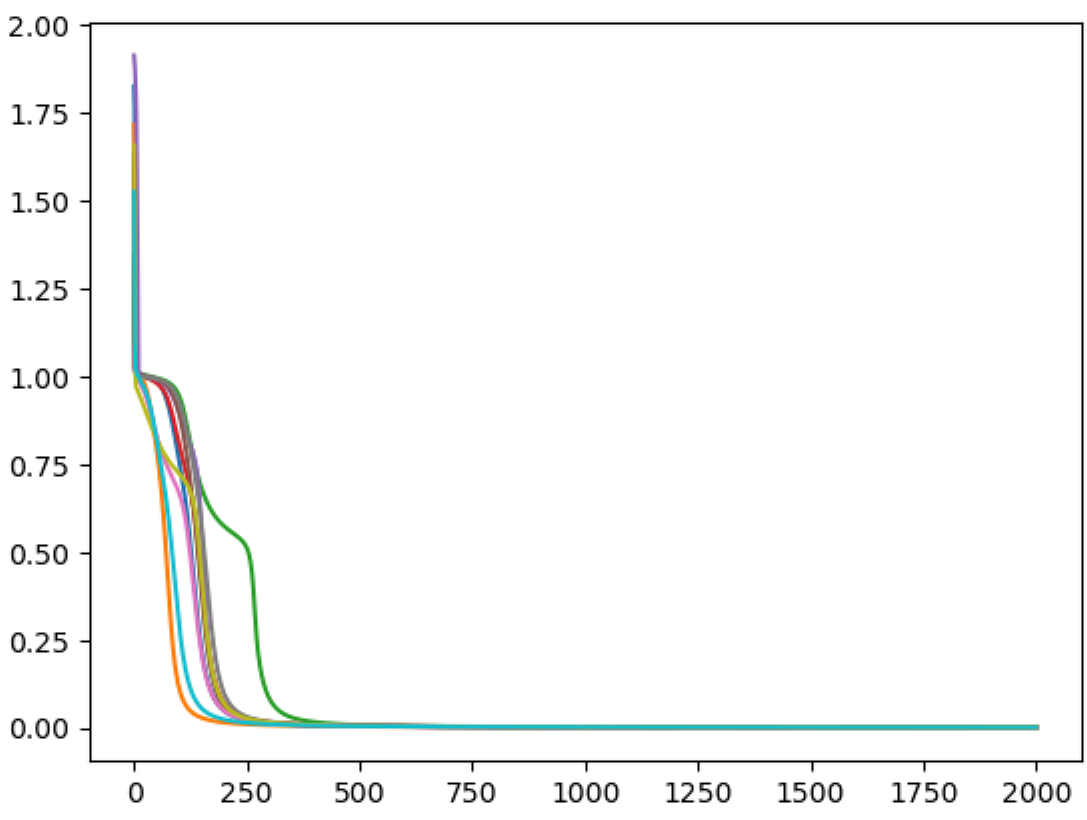
In [1]:
```python
import numpy as np
import matplotlib.pyplot as plt
import itertools

num_of_iter=2000
eta=0.1 # Learning rate
x_u_list = np.array([[0,0],[0,1],[1,0],[1,1]])
y_u_list = np.array([[1],[-1],[-1],[1]])
input_dim, number_of_neuron_hidden_layer, number_of_output = 2,2,1

def g_function(x):
    """Activation function

    Function:
            g(h) = tanh (beta * h)
    with beta = 1
            g'(g)= beta * (1-g ^2 )

    Args:
        x (float): input value

    Returns:
        float: output value
    """
    return np.tanh(x)
```

In [2]:
```python
number_of_trials = 10 # Number of time we want to run the training process w
error = np.zeros([ number_of_trials , num_of_iter])

for idx_num_of_trial in range(number_of_trials):
    # 1. Initialize the weights to small random values. (layer 1)
    layer1_wjk = np.random.uniform(size=(number_of_neuron_hidden_layer,input
    layer1_bias = np.random.uniform(size=(1,number_of_neuron_hidden_layer))

    # 1. Initialize the weights to small random values. (layer 2)
    layer2_wij = np.random.uniform(size=(number_of_output,number_of_neuron_h
    layer2_bias = np.random.uniform(size=(1,number_of_output))

    for i in range(num_of_iter):
        layer1_hj = np.dot(x_u_list,layer1_wjk.T)  #Activation h_j= x*w_jk
        layer1_hj += layer1_bias           #Bias

        # Propagate the signal forwards through the network
        layer1_Vj = g_function(layer1_hj)       #Activation function g(x) ap

        layer2_hi = np.dot(layer1_Vj,layer2_wij.T)
        layer2_hi += layer2_bias
        layer2_Vi = g_function(layer2_hi) #Activation function g(x) applied

        # Compute the delta for the output layer
        layer2_delta_i = (1-layer2_Vi**2) * (y_u_list-layer2_Vi) # g'(h_i) *

        layer2_delta_bias = eta*np.sum(layer2_delta_i,0)
```

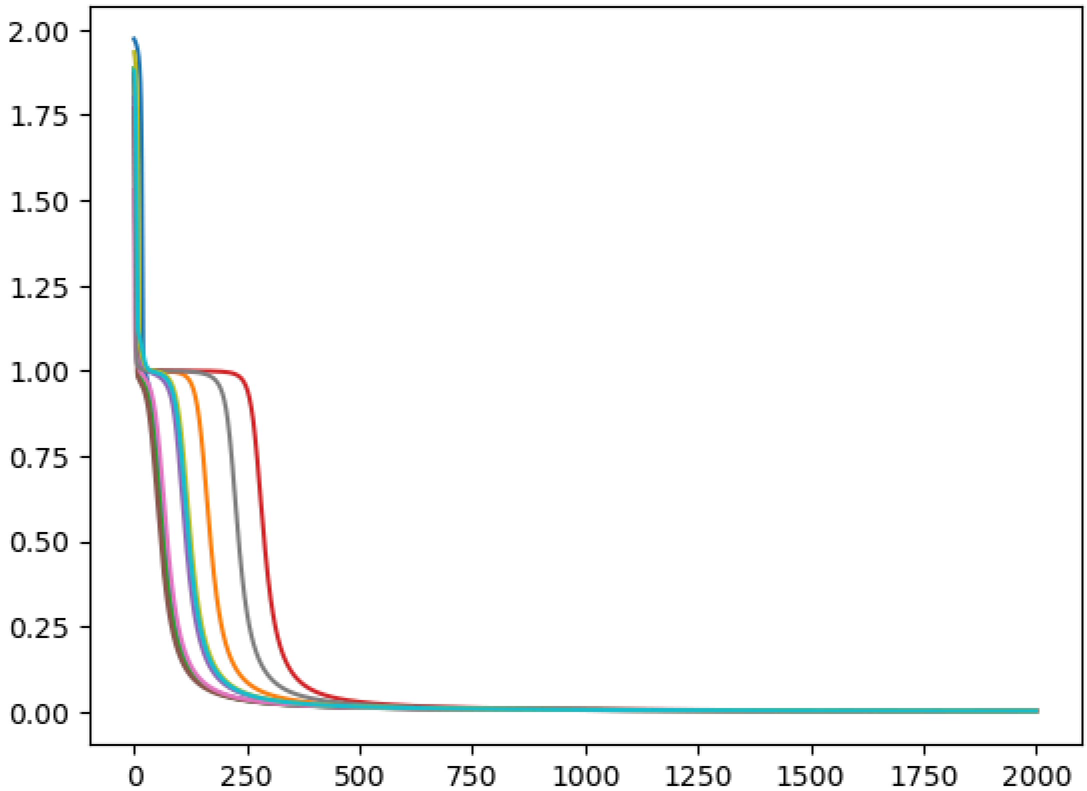
```python
        # Compute the delta for the hidden layer
        layer1_delta_j = (1-layer1_Vj**2)*(layer2_delta_i.dot(layer2_wij))
        layer1_delta_bias = eta*np.sum(layer1_delta_j,0)

        # 6. Compute the gradient of the error with respect to the weights
        layer2_delta_Wij = eta*layer2_delta_i.T.dot(layer1_Vj)
        layer1_delta_wjk = eta*(layer1_delta_j.T).dot(x_u_list)

        # 6.1 Update the weights
        layer1_wjk += layer1_delta_wjk
        layer1_bias += layer1_delta_bias
        layer2_wij += layer2_delta_Wij
        layer2_bias += layer2_delta_bias

        # 7. Compute the error
        error[idx_num_of_trial,i] = np.sum((y_u_list-layer2_Vi)**2)/4

# fig =plt.subplots(figsize=(15,5))
# plt.plot(sum(error,1))

for i_idx in range(number_of_trials):
   plt.plot(error[i_idx,:])
plt.show()
```



Parte B

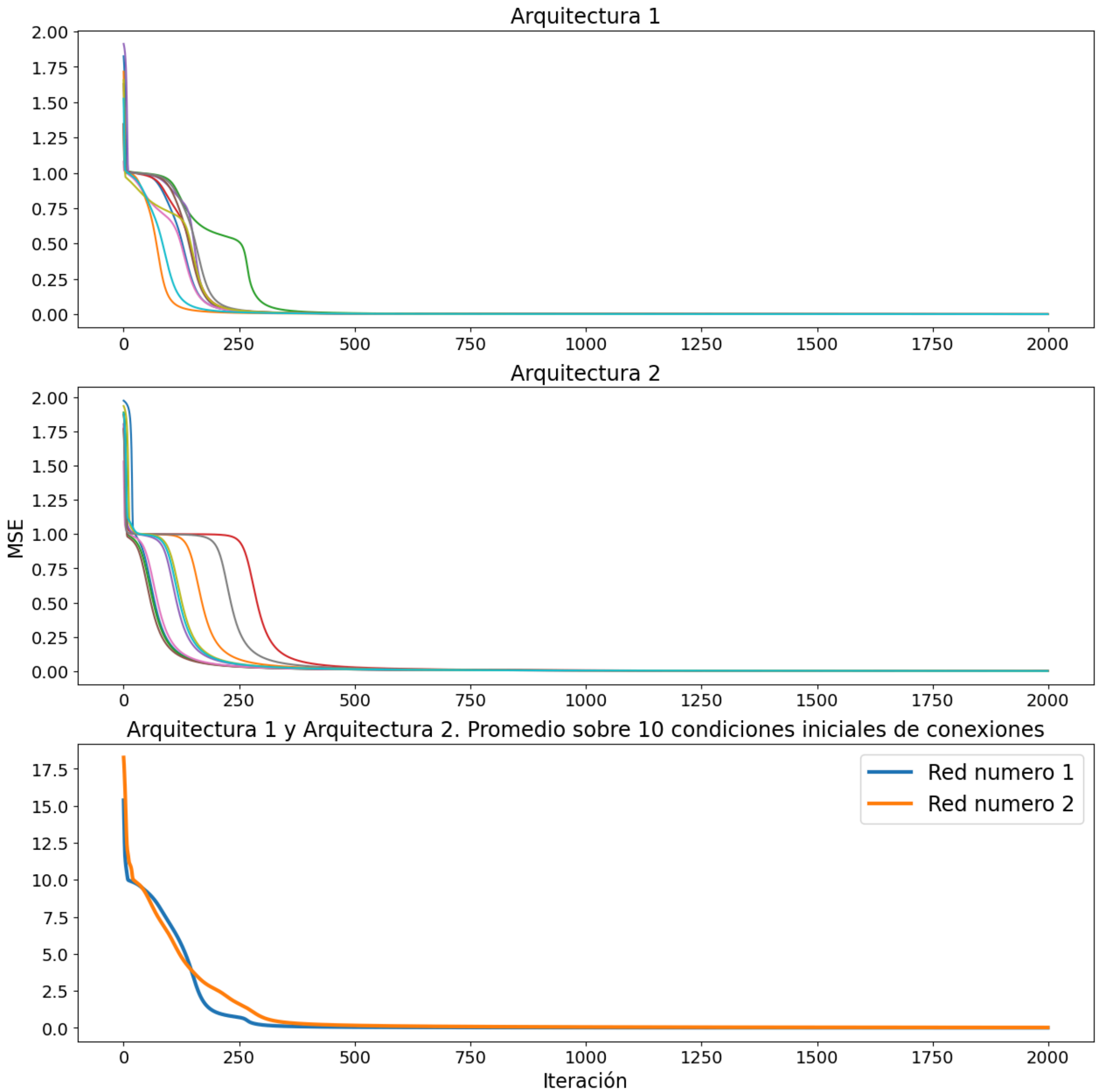
In [3]:
```python
number_of_trials = 10
num_of_iterations=2000
error_model_2 = np.zeros([number_of_trials,num_of_iterations])
for i_idx in range(number_of_trials):
```

```python
    layer1_wjk = np.random.uniform(size=(1,2))
    layer1_bias = np.random.uniform(size=(1,1))

    layer2_Wij = np.random.uniform(size=(1,3)) # The first two are for the i
    layer2_bias = np.random.uniform(size=(1,1))

    for i in range(num_of_iterations):
        layer1_hj = np.dot(x_u_list,layer1_wjk.T)           #Aca hago la su
        layer1_hj += layer1_bias                           # Assumed that bias
        layer1_Vj = g_function(layer1_hj)

        layer2_hi = np.dot( np.concatenate( [x_u_list, layer1_Vj],axis=1) ,l
        layer2_hi += layer2_bias
        layer2_Oi = g_function(layer2_hi)

        # 4. Compute the delta for the output layer
        layer2_delta_i = (1-layer2_Oi**2) * (y_u_list-layer2_Oi) # g'(h_i) *
        layer2_delta_bias = eta*np.sum(layer2_delta_i,0)

        # 5 Compute the delta for the hidden layers
        layer1_delta_j = (1-layer1_Vj**2)*(layer2_delta_i.dot(layer2_Wij[0,2
        layer1_delta_bias = eta*np.sum(layer1_delta_j,0)

        # 6. Compute the gradient of the error with respect to the weights
        layer2_delta_Wij = eta*layer2_delta_i.T.dot(np.concatenate([x_u_list
        layer1_delta_wjk = eta*(layer1_delta_j.T).dot(x_u_list)

        # 6.1 Update the weights
        layer1_wjk += layer1_delta_wjk
        layer1_bias += layer1_delta_bias
        layer2_Wij += layer2_delta_Wij
        layer2_bias += layer2_delta_bias

        # 7. Compute the error
        error_model_2[i_idx,i] = np.sum((y_u_list-layer2_Oi)**2)/4

# fig =plt.subplots(figsize=(15,5))
# plt.plot(sum(error_model_2,0))


for i_idx in range(number_of_trials):
   plt.plot(error_model_2[i_idx,:])
plt.show()
```



In [4]:
```
fig , axes =plt.subplots(figsize=(15,15))
axes.axis('off')

plt.subplot(311)
plt.title('Arquitectura 1',fontsize='xx-large')

for i_idx in range(number_of_trials):
    plt.plot(error[i_idx,:])
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)

plt.subplot(312)
plt.title('Arquitectura 2',fontsize='xx-large')

for i_idx in range(number_of_trials):
    plt.plot(error_model_2[i_idx,:])
plt.ylabel('MSE',fontsize=16)
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)

plt.subplot(313)
plt.title('Arquitectura 1 y Arquitectura 2. Promedio sobre 10 condiciones ir
plt.plot(sum(error,0),linewidth=3)
plt.plot(sum(error_model_2,0),linewidth=3)
plt.legend(['Red numero 1','Red numero 2'],fontsize='xx-large')
plt.xlabel('Iteración',fontsize=16)
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
```

```
Out[4]: (array([-2.5,  0. ,  2.5,  5. ,  7.5, 10. , 12.5, 15. , 17.5, 20. ]),
         [Text(0, -2.5, '-2.5'),
          Text(0, 0.0, '0.0'),
          Text(0, 2.5, '2.5'),
          Text(0, 5.0, '5.0'),
          Text(0, 7.5, '7.5'),
          Text(0, 10.0, '10.0'),
          Text(0, 12.5, '12.5'),
          Text(0, 15.0, '15.0'),
          Text(0, 17.5, '17.5'),
          Text(0, 20.0, '20.0')])
```

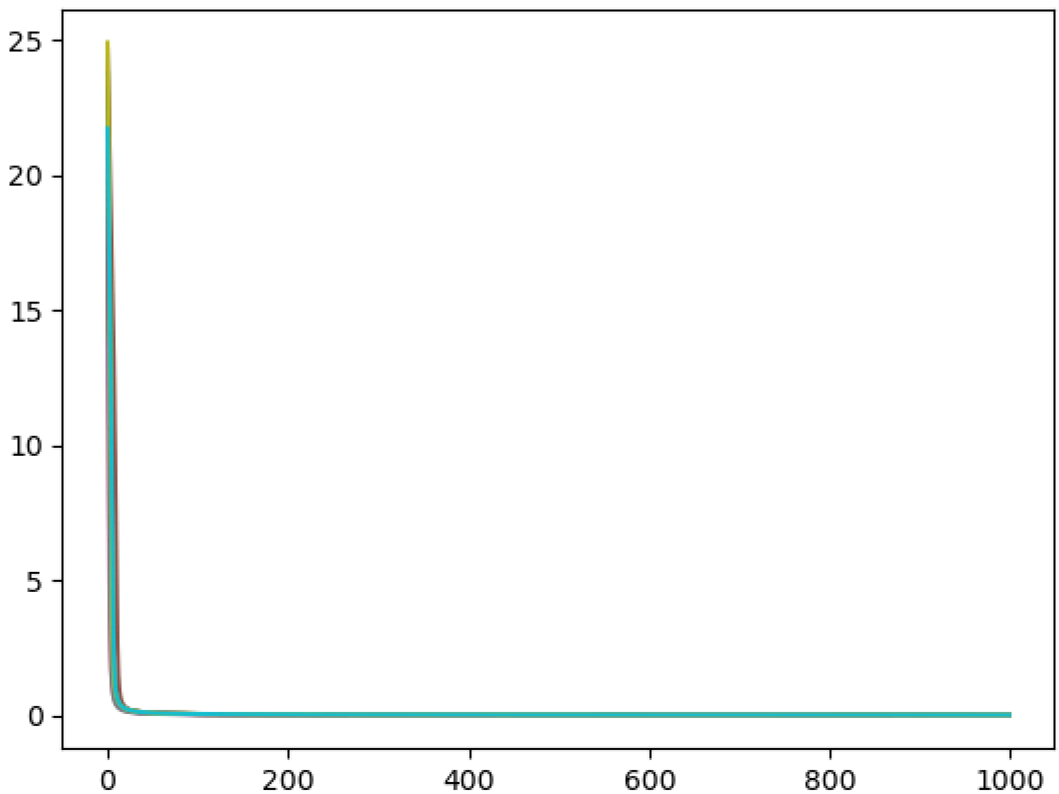


Problema 2

```
In [5]: num_of_iter = 1000
        eta = 0.01 # Learning rate
        N = 5 # Number of model input
```

```
# Sample data
# Generate all possible combinations of N inputs, each input being either +1
x_u_list = list(itertools.product([1, -1], repeat=N))

for x_sample in x_u_list:
        # The output y is +1 if the product of inputs is +1, otherwise -1
        y_u_list = 1 if all(x_i == 1 for x_i in x_sample) else -1 if all(x_i


# Model
input_dim = N
number_of_neuron_hidden_layer = 2
number_of_output = 1

neuron_counts = [1, 3, 5, 7, 9, 11]  # Neuron counts for hidden layer

number_of_trials = 10 # Number of time we want to run the training process w
error = np.zeros([ number_of_trials , num_of_iter])


for idx_num_of_trial in range(number_of_trials):
    # 1. Initialize the weights to small random values. (layer 1)
    layer1_wjk = np.random.uniform(size=(number_of_neuron_hidden_layer,input
    layer1_bias = np.random.uniform(size=(1,number_of_neuron_hidden_layer))

    # 1. Initialize the weights to small random values. (layer 2)
    layer2_wij = np.random.uniform(size=(number_of_output,number_of_neuron_h
    layer2_bias = np.random.uniform(size=(1,number_of_output))

    for i in range(num_of_iter):
        layer1_hj = np.dot(x_u_list,layer1_wjk.T)  #Activation h_j= x*w_jk
        layer1_hj += layer1_bias           #Bias

        # Propagate the signal forwards through the network
        layer1_Vj = g_function(layer1_hj)       #Activation function g(x) ap

        layer2_hi = np.dot(layer1_Vj,layer2_wij.T)
        layer2_hi += layer2_bias
        layer2_Vi = g_function(layer2_hi) #Activation function g(x) applied

        # Compute the delta for the output layer
        layer2_delta_i = (1-layer2_Vi**2) * (y_u_list-layer2_Vi) # g'(h_i) *

        layer2_delta_bias = eta*np.sum(layer2_delta_i,0)

        # Compute the delta for the hidden layer
        layer1_delta_j = (1-layer1_Vj**2)*(layer2_delta_i.dot(layer2_wij))
        layer1_delta_bias = eta*np.sum(layer1_delta_j,0)

        # 6. Compute the gradient of the error with respect to the weights
        layer2_delta_Wij = eta*layer2_delta_i.T.dot(layer1_Vj)
        layer1_delta_wjk = eta*(layer1_delta_j.T).dot(x_u_list)

        # 6.1 Update the weights
        layer1_wjk += layer1_delta_wjk
        layer1_bias += layer1_delta_bias
```

```
        layer2_wij += layer2_delta_Wij
        layer2_bias += layer2_delta_bias

        # 7. Compute the error
        error[idx_num_of_trial,i] = np.sum((y_u_list-layer2_Vi)**2)/4

# fig =plt.subplots(figsize=(15,5))
# plt.plot(sum(error,1))

for i_idx in range(number_of_trials):
   plt.plot(error[i_idx,:])
plt.show()
```



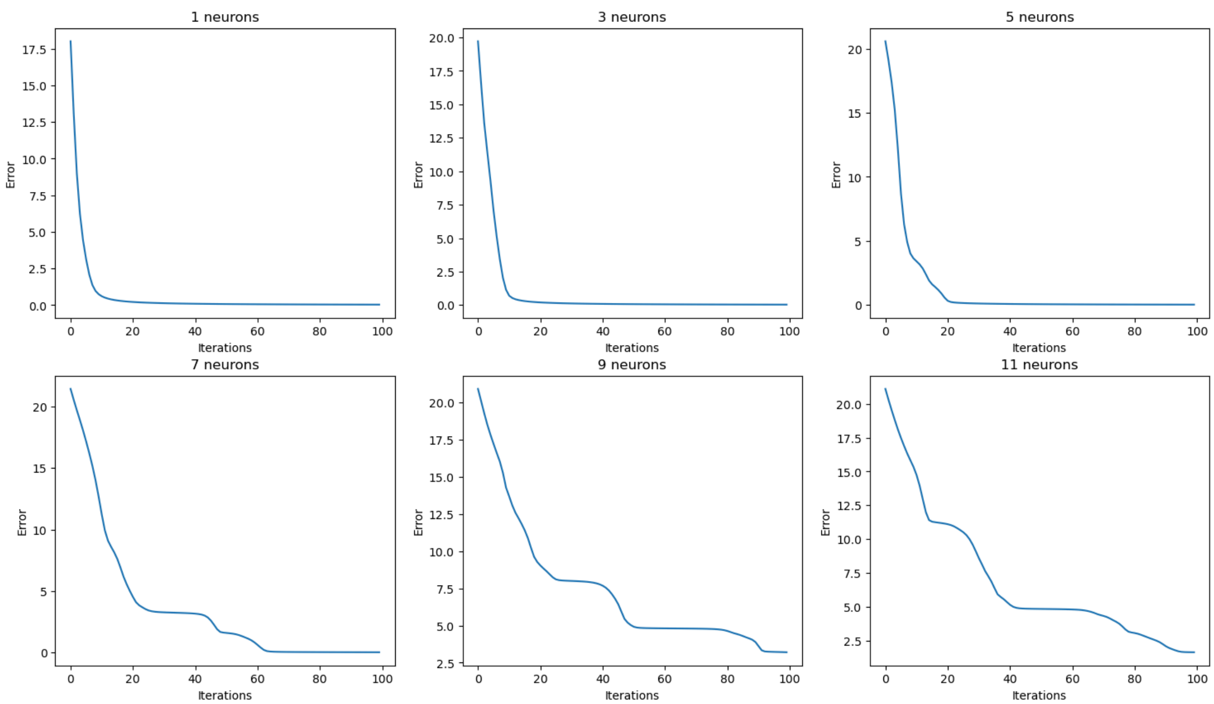
For multiple number of neurons in the hidden layer

In [6]:

```
num_of_iter = 100
eta = 0.01 # Learning rate
N = 5 # Number of model input

# Sample data
# Generate all possible combinations of N inputs, each input being either +1
x_u_list = list(itertools.product([1, -1], repeat=N))

for x_sample in x_u_list:
        # The output y is +1 if the product of inputs is +1, otherwise -1
        y_u_list = 1 if all(x_i == 1 for x_i in x_sample) else -1 if all(x_i


# Model
input_dim = N
number_of_neuron_hidden_layer = 11
```

```python
number_of_output = 1

neuron_counts = [1, 3, 5, 7, 9, 11]  # Neuron counts for hidden layer

number_of_trials = 10 # Number of time we want to run the training process w
error = np.zeros([ number_of_trials , num_of_iter])

# Set up the plot
fig, axes = plt.subplots(2, 3, figsize=(18, 10))
axes = axes.ravel()  # Flatten axes for easy indexing

# Loop over different numbers of neurons in the hidden layer
for idx, number_of_neuron_hidden_layer in enumerate(neuron_counts):

    for idx_num_of_trial in range(number_of_trials):
        # 1. Initialize the weights to small random values. (layer 1)
        layer1_wjk = np.random.uniform(size=(number_of_neuron_hidden_layer,i
        layer1_bias = np.random.uniform(size=(1,number_of_neuron_hidden_laye

        # 1. Initialize the weights to small random values. (layer 2)
        layer2_wij = np.random.uniform(size=(number_of_output,number_of_neur
        layer2_bias = np.random.uniform(size=(1,number_of_output))

        for i in range(num_of_iter):
            layer1_hj = np.dot(x_u_list,layer1_wjk.T)  #Activation h_j= x*w_
            layer1_hj += layer1_bias           #Bias

            # Propagate the signal forwards through the network
            layer1_Vj = g_function(layer1_hj)      #Activation function g(x

            layer2_hi = np.dot(layer1_Vj,layer2_wij.T)
            layer2_hi += layer2_bias
            layer2_Vi = g_function(layer2_hi) #Activation function g(x) appl

            # Compute the delta for the output layer
            layer2_delta_i = (1-layer2_Vi**2) * (y_u_list-layer2_Vi) # g'(h_

            layer2_delta_bias = eta*np.sum(layer2_delta_i,0)

            # Compute the delta for the hidden layer
            layer1_delta_j = (1-layer1_Vj**2)*(layer2_delta_i.dot(layer2_wij
            layer1_delta_bias = eta*np.sum(layer1_delta_j,0)

            # 6. Compute the gradient of the error with respect to the weigh
            layer2_delta_Wij = eta*layer2_delta_i.T.dot(layer1_Vj)
            layer1_delta_wjk = eta*(layer1_delta_j.T).dot(x_u_list)

            # 6.1 Update the weights
            layer1_wjk += layer1_delta_wjk
            layer1_bias += layer1_delta_bias
            layer2_wij += layer2_delta_Wij
            layer2_bias += layer2_delta_bias

            # 7. Compute the error
            error[idx_num_of_trial,i] = np.sum((y_u_list-layer2_Vi)**2)/4
```

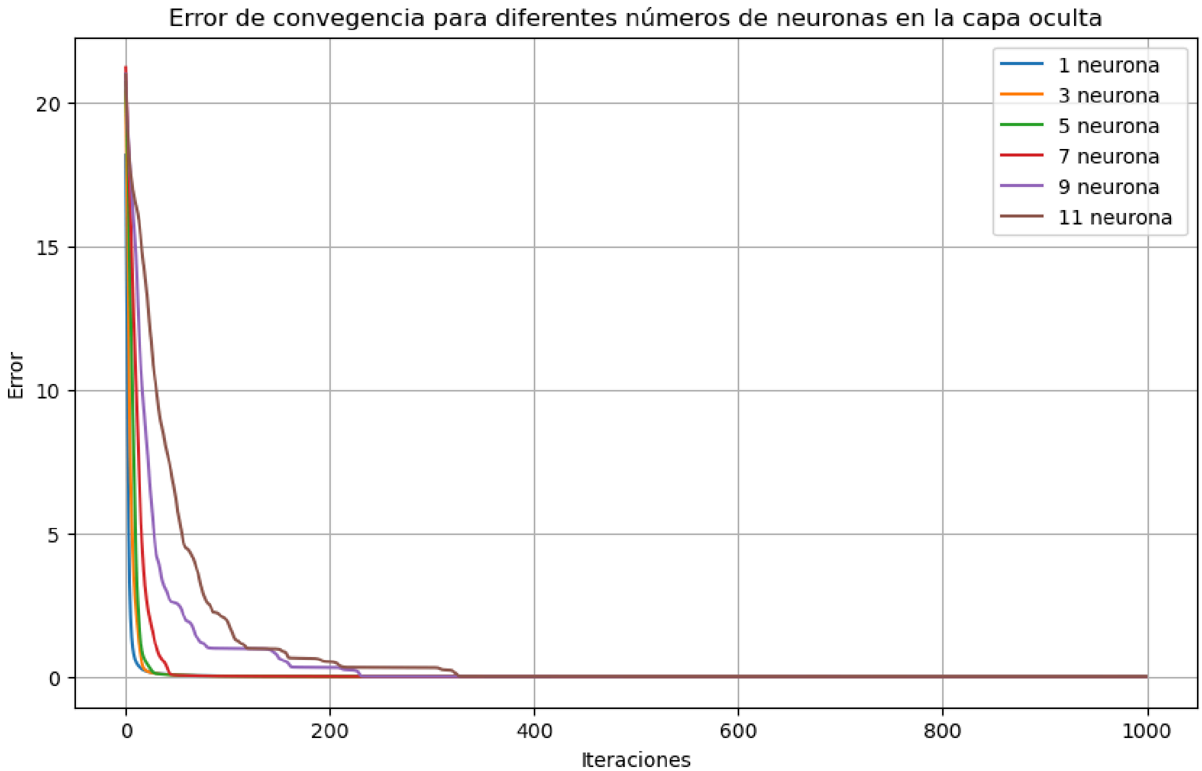
```python
# Plot the average error over all trials for this number of neurons
    avg_error = np.mean(error, axis=0)
    ax = axes[idx]
    ax.plot(avg_error)
    ax.set_title(f"{number_of_neuron_hidden_layer} neurons")
    ax.set_xlabel("Iterations")
    ax.set_ylabel("Error")

plt.show()
```



In [7]:
```python
num_of_iter = 1000
eta = 0.01 # Learning rate
N = 5 # Number of model input

# Sample data
# Generate all possible combinations of N inputs, each input being either +1
x_u_list = list(itertools.product([1, -1], repeat=N))

for x_sample in x_u_list:
        # The output y is +1 if the product of inputs is +1, otherwise -1
        y_u_list = 1 if all(x_i == 1 for x_i in x_sample) else -1 if all(x_i


# Model
input_dim = N
number_of_neuron_hidden_layer = 11
number_of_output = 1

neuron_counts = [1, 3, 5, 7, 9, 11]  # Neuron counts for hidden layer

number_of_trials = 50 # Number of time we want to run the training process w
error = np.zeros([ number_of_trials , num_of_iter])
```

```python
# Set up the plot
plt.figure(figsize=(10, 6))

# Loop over different numbers of neurons in the hidden layer
for idx, number_of_neuron_hidden_layer in enumerate(neuron_counts):

    for idx_num_of_trial in range(number_of_trials):
        # 1. Initialize the weights to small random values. (layer 1)
        layer1_wjk = np.random.uniform(size=(number_of_neuron_hidden_layer,i
        layer1_bias = np.random.uniform(size=(1,number_of_neuron_hidden_laye

        # 1. Initialize the weights to small random values. (layer 2)
        layer2_wij = np.random.uniform(size=(number_of_output,number_of_neur
        layer2_bias = np.random.uniform(size=(1,number_of_output))

        for i in range(num_of_iter):
            layer1_hj = np.dot(x_u_list,layer1_wjk.T)  #Activation h_j= x*w_
            layer1_hj += layer1_bias           #Bias

            # Propagate the signal forwards through the network
            layer1_Vj = g_function(layer1_hj)       #Activation function g(x

            layer2_hi = np.dot(layer1_Vj,layer2_wij.T)
            layer2_hi += layer2_bias
            layer2_Vi = g_function(layer2_hi) #Activation function g(x) appl

            # Compute the delta for the output layer
            layer2_delta_i = (1-layer2_Vi**2) * (y_u_list-layer2_Vi) # g'(h_

            layer2_delta_bias = eta*np.sum(layer2_delta_i,0)

            # Compute the delta for the hidden layer
            layer1_delta_j = (1-layer1_Vj**2)*(layer2_delta_i.dot(layer2_wij
            layer1_delta_bias = eta*np.sum(layer1_delta_j,0)

            # 6. Compute the gradient of the error with respect to the weigh
            layer2_delta_Wij = eta*layer2_delta_i.T.dot(layer1_Vj)
            layer1_delta_wjk = eta*(layer1_delta_j.T).dot(x_u_list)

            # 6.1 Update the weights
            layer1_wjk += layer1_delta_wjk
            layer1_bias += layer1_delta_bias
            layer2_wij += layer2_delta_Wij
            layer2_bias += layer2_delta_bias

            # 7. Compute the error
            error[idx_num_of_trial,i] = np.sum((y_u_list-layer2_Vi)**2)/4



    # Plot the average error over all trials for this number of neurons
    avg_error = np.mean(error, axis=0)
    plt.plot(avg_error, label=f'{number_of_neuron_hidden_layer} neurona ')

# Add labels and legend to the plot
```

```
plt.title("Error de convegencia para diferentes números de neuronas en la ca
plt.xlabel("Iteraciones")
plt.ylabel("Error")
plt.legend()
plt.grid(True)

# Display the plot
plt.show()
```



Problema 3

In [16]:
```
"""
Modeling a logistic regression using a neural network with back-propagation

"""
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split

# Set seed for reproducibility
seed=43
np.random.seed(seed)
tf.random.set_seed(seed)




def create_model():
    # Network architecture
```

```python
    input_dim=1                                    # Number of neurons in the input
    hidden_dim=5                                    # Number of neurons in the hidde
    output_dim=1                                    # Number of neurons in the outpu

    # ------------------------ Model----------------------------------------
    # Input layer
    input_layer=tf.keras.layers.Input(shape=(input_dim,), name='Input')
    # Hidden layer - Sigmoid activation function - Name Hidden_layer
    hidden_layer=tf.keras.layers.Dense(hidden_dim, activation='sigmoid', nam

    # Concatenate input and hidden layer
    concatenated_layer=tf.keras.layers.Concatenate(name='Concate')([input_la

    # Output layer - Linear activation function
    output_layer=tf.keras.layers.Dense(output_dim, activation='linear', name

    # Build the model
    model=tf.keras.Model(inputs=input_layer, outputs=output_layer)

    # Optimizer
    opt=tf.keras.optimizers.RMSprop(learning_rate=0.01)

    # Compile the model
    model.compile(optimizer=opt, loss='MSE')
    return model

# Create the model
model=create_model()

# Summary of the model
model.summary()

# Plot the model
tf.keras.utils.plot_model(model, to_file='model.png', show_shapes=False, sho



n_examples_list = [5, 10, 100]

for n_examples in n_examples_list:
    # ---------------- Input data---------------------------------
    # Generate random data list for x
    x = np.random.uniform(low=0,high=1 ,size = n_examples).reshape(-1,1)   #
    print(f"valore x: {x}")



    # Generate y data
    y = 4 * x * (1 - x)
    print(f"valore y: {y}")

    x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2,

    # Train the model
    history=model.fit(x=x_train, y=y_train,
                      epochs=1500,
```

```
                    batch_size=4,
                    shuffle=False,
                    validation_data=(x_test, y_test),
                    verbose=False)


    # ---------------- Evaluate the model--------------------------------
    # Evaluate the model on the train set
    loss_train = model.evaluate(x_train, y_train, verbose=0)

    print(f"Example with {n_examples} Samples - Train loss {loss_train}")

    # Evaluate the model on the test set
    loss_test = model.evaluate(x_test, y_test, verbose=0)

    print(f"Example with {n_examples} Samples - Test loss {loss_test}")

    # Make predictions for 0 - 1 values
    x_predict = np.linspace(0, 1, 100).reshape(-1,1)
    y_predict = model.predict(x_predict)

    loss_predict = model.evaluate(x_predict, 4 * x_predict * (1 - x_predict)

    print(f"Example with {n_examples} Samples - Predict loss {loss_predict}"

    # Save the predictions in file for plotting
    np.savetxt(f'lr-predictions_{n_examples}.txt', np.concatenate((x_predict

    # Save error history values for plotting
    np.savetxt(f'lr-error_{n_examples}.txt', np.array([history.history['loss







# encoded_log = model.predict(x_test, verbose=True)
# print(encoded_log.shape)




# # "Loss"
# plt.plot(np.sqrt(history.history['loss']))
# plt.plot(np.sqrt(history.history['val_loss']))
# plt.title('model loss')
# plt.ylabel('loss')
# plt.xlabel('epoch')
# plt.legend(['train', 'validation'], loc='upper left')
# plt.show()
```

Model: "functional_5"

Layer (type)	Output Shape	Param #	Connected to
Input (InputLayer)	(None, 1)	0	-
Hidden_layer (Dense)	(None, 5)	10	Input[0][0]
Concate (Concatenate)	(None, 6)	0	Input[0][0], Hidden_layer[
Output (Dense)	(None, 1)	7	Concate[0][0]

Total params: 17 (68.00 B)

Trainable params: 17 (68.00 B)

Non-trainable params: 0 (0.00 B)

```
You must install pydot (`pip install pydot`) for `plot_model` to work.
valore x: [[0.11505457]
 [0.60906654]
 [0.13339096]
 [0.24058962]
 [0.32713906]]
valore y: [[0.40726805]
 [0.95241796]
 [0.46239126]
 [0.73082502]
 [0.88047638]]
Example with 5 Samples - Train loss 0.0023474260233342648
Example with 5 Samples - Test loss 0.0046743000857532024
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
Example with 5 Samples - Predict loss 0.09869936108589172
valore x: [[0.85913749]
 [0.66609021]
 [0.54116221]
 [0.02901382]
 [0.7337483 ]
 [0.39495002]
 [0.80204712]
 [0.25442113]
 [0.05688494]
 [0.86664864]]
valore y: [[0.48408105]
 [0.88965616]
 [0.99322269]
 [0.11268809]
 [0.78144694]
 [0.95585801]
 [0.63507015]
 [0.75876407]
 [0.21459616]
 [0.4622751 ]]
Example with 10 Samples - Train loss 0.0011201611487194896
Example with 10 Samples - Test loss 0.0036995145492255688
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 838us/step
Example with 10 Samples - Predict loss 0.005022585857659578
valore x: [[0.221029  ]
 [0.40498945]
 [0.31609647]
 [0.0766627 ]
 [0.84322469]
 [0.84893915]
 [0.97146509]
 [0.38537691]
 [0.95448813]
 [0.44575836]
 [0.66972465]
 [0.08250005]
 [0.89709858]
 [0.2980035 ]
 [0.26230482]
 [0.00512955]
 [0.54320252]
```

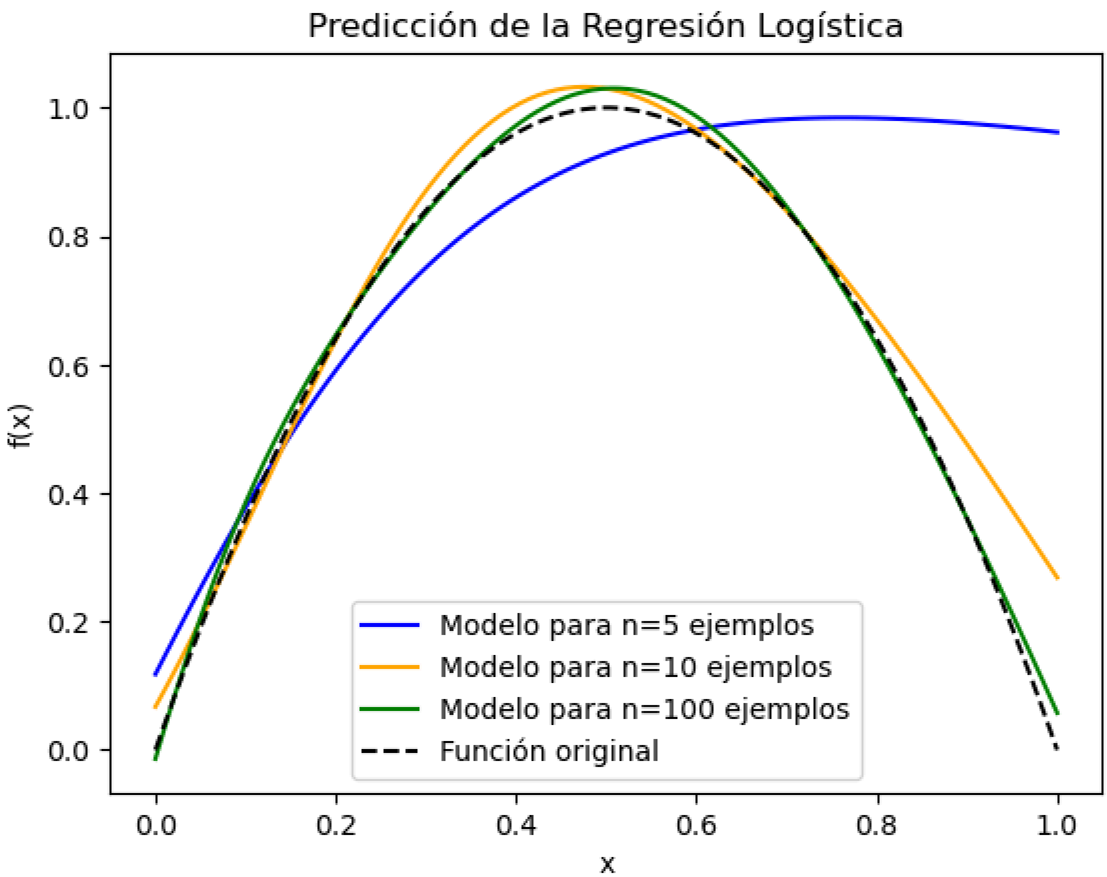
```
[0.47559637]
[0.63637368]
[0.97820413]
[0.90866276]
[0.91015308]
[0.52525567]
[0.10401895]
[0.1809146 ]
[0.95304022]
[0.41195298]
[0.86501712]
[0.67217728]
[0.6287858 ]
[0.27555878]
[0.89674727]
[0.20689137]
[0.40440524]
[0.99357249]
[0.73572708]
[0.44506141]
[0.56066317]
[0.41125549]
[0.72698799]
[0.39919689]
[0.67014516]
[0.70471561]
[0.60955987]
[0.54003446]
[0.20608185]
[0.19916148]
[0.79573886]
[0.29033278]
[0.65596283]
[0.29961678]
[0.14447838]
[0.40395667]
[0.31026953]
[0.24339802]
[0.58810404]
[0.24534325]
[0.74777061]
[0.72014665]
[0.69526087]
[0.10274278]
[0.94364243]
[0.50333963]
[0.89967362]
[0.19857988]
[0.59444919]
[0.96540858]
[0.99869825]
[0.02416862]
[0.48130333]
[0.29142269]
[0.06372057]
[0.5696244 ]
```

```
 [0.00508328]
 [0.61127759]
 [0.87018148]
 [0.88360146]
 [0.95431948]
 [0.73986382]
 [0.18471296]
 [0.43467832]
 [0.8858995 ]
 [0.25504628]
 [0.44331269]
 [0.61693698]
 [0.10335251]
 [0.49010966]
 [0.0447044 ]
 [0.31162747]
 [0.75203969]
 [0.71549909]
 [0.94155424]
 [0.77689537]
 [0.21522851]
 [0.90497118]
 [0.55226415]
 [0.84489789]
 [0.92248948]
 [0.82896128]
 [0.394198  ]]
valore y: [[0.68870072]
 [0.96389198]
 [0.86471797]
 [0.28314212]
 [0.52878724]
 [0.51296588]
 [0.11088267]
 [0.94744619]
 [0.17376218]
 [0.98823138]
 [0.88477417]
 [0.30277517]
 [0.36925087]
 [0.83678966]
 [0.77400401]
 [0.02041295]
 [0.99253417]
 [0.99761785]
 [0.92560888]
 [0.08528322]
 [0.331979  ]
 [0.32709782]
 [0.9974486 ]
 [0.37279603]
 [0.59273802]
 [0.17901823]
 [0.96899089]
 [0.46705001]
 [0.88141994]
```

```
[0.93365688]
[0.79850455]
[0.37036642]
[0.65634933]
[0.96344657]
[0.02554478]
[0.77773097]
[0.98792701]
[0.98527992]
[0.96849765]
[0.79390581]
[0.95935493]
[0.88420249]
[0.83236608]
[0.95198654]
[0.99358897]
[0.65444849]
[0.63798473]
[0.6501541 ]
[0.82415862]
[0.90270239]
[0.83938625]
[0.49441751]
[0.96310271]
[0.85600939]
[0.73662169]
[0.96895071]
[0.74059976]
[0.7544389 ]
[0.80614181]
[0.84749278]
[0.3687468 ]
[0.21272558]
[0.99995539]
[0.36104399]
[0.63658366]
[0.9643174 ]
[0.13357941]
[0.00520024]
[0.09433801]
[0.99860174]
[0.82598202]
[0.23864103]
[0.98060977]
[0.02022978]
[0.95046919]
[0.4518627 ]
[0.41139966]
[0.17437523]
[0.76986139]
[0.60237632]
[0.98293231]
[0.40432632]
[0.7599907 ]
[0.9871462 ]
[0.94530297]
```

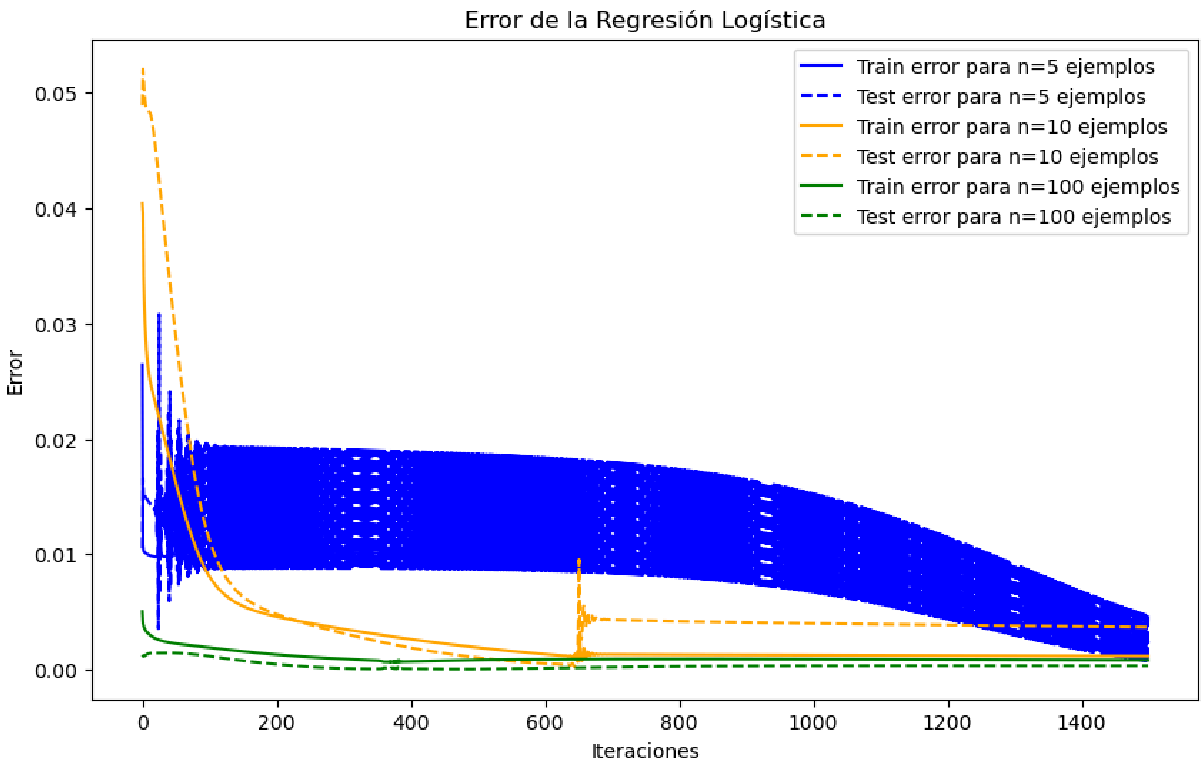
```
 [0.37068308]
 [0.99960872]
 [0.17082368]
 [0.85806316]
 [0.74590399]
 [0.81424057]
 [0.2201194 ]
 [0.69331582]
 [0.67562081]
 [0.34399337]
 [0.98907384]
 [0.52418179]
 [0.28601054]
 [0.56713791]
 [0.95522375]]
Example with 100 Samples - Train loss 0.00035510802990756 93
Example with 100 Samples - Test loss 0.0003401233989279717
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 642us/step
Example with 100 Samples - Predict loss 0.00035896486951969564
```

In [17]:
```python
# List of colors
colors = ['blue', 'orange', 'green']

# Open the file for n=5,10,100 and read the data
for n_examples, color in zip(n_examples_list, colors):
    # Load
    data = np.loadtxt(f'lr-predictions_{n_examples}.txt')
    x_predict = data[:,0]
    y_predict = data[:,1]

    # Plot the data
    plt.plot(x_predict, y_predict, color= color, label=f'Modelo para n={n_ex

# Plot the original function
x = np.linspace(0, 1, 100)
y = 4 * x * (1 - x)
plt.plot(x, y, "--", color='black', label='Función original')


# Add labels and legend
plt.xlabel('x')
plt.ylabel('f(x)')
plt.legend()
plt.title('Predicción de la Regresión Logística')
plt.show()
```



In [18]:

```
# Open error files for n=5,10,100 and read the data

# Set up the plot
plt.figure(figsize=(10, 6))
# List of colors
colors = ['blue', 'orange', 'green']

for n_examples,color in zip(n_examples_list,colors):
    # Load
    data = np.loadtxt(f'lr-error_{n_examples}.txt')
    train_error = data[:,0]
    test_error = data[:,1]

    # Plot the data. Train is solid and test is dashed
    plt.plot(train_error, "-",color = color ,label=f'Train error para n={n_e
    plt.plot(test_error, "--",color = color ,label=f'Test error para n={n_ex

# Add labels and legend
plt.xlabel('Iteraciones')
plt.ylabel('Error')
plt.legend()
plt.title('Error de la Regresión Logística')
plt.show()
```



In [14]:
```
# ######################################################################
# # Output files
# fout=open("lr-out.dat","wb")
# ftrain=open("lr-train.dat","wb")
# ftest=open("lr-test.dat","wb")
# #
# np.savetxt(ftrain,np.c_[x_train,y_train],delimiter=" ")
# np.savetxt(ftest,np.c_[x_test, y_test],delimiter=" ")
# np.savetxt(fout,np.c_[x_test, encoded_log],delimiter=" ")

# W_Input_Hidden = model.layers[0].get_weights()
# W_Output_Hidden = model.layers[1].get_weights()
# # B_Input_Hidden = model.layers[0].get_weights()[1]
# # B_Output_Hidden = model.layers[1].get_weights()[1]
# #print(summary)
# print('INPUT-HIDDEN LAYER WEIGHTS:')
# print(W_Input_Hidden)
# print('HIDDEN-OUTPUT LAYER WEIGHTS:')
# print(W_Output_Hidden)

# # print('INPUT-HIDDEN LAYER BIAS:')
# # print(B_Input_Hidden)
# # print('HIDDEN-OUTPUT LAYER BIAS:')
# # print(B_Output_Hidden)
```

```
INPUT-HIDDEN LAYER WEIGHTS:
[]
HIDDEN-OUTPUT LAYER WEIGHTS:
[array([[ 0.41649264, -0.88370305,  0.7433394 ,  0.22161928,  0.03575625]],
      dtype=float32), array([ 0.0649383 , -0.09769494,  0.02043273,  0.02721
332, -0.02572968],
      dtype=float32)]
```