Labs 11

Smain Belghazi ING3

In [2]:

```
%pip install neurolab
%pip install matplotlib
```

```
Requirement already satisfied: neurolab in c:\users\smain\appdata\local\programs
\python\python312\lib\site-packages (0.3.5)
Note: you may need to restart the kernel to use updated packages.
Collecting matplotlib
  Downloading matplotlib-3.10.0-cp312-cp312-win_amd64.whl.metadata (11 kB)
Collecting contourpy>=1.0.1 (from matplotlib)
  Downloading contourpy-1.3.1-cp312-cp312-win_amd64.whl.metadata (5.4 kB)
Collecting cycler>=0.10 (from matplotlib)
  Using cached cycler-0.12.1-py3-none-any.whl.metadata (3.8 kB)
Collecting fonttools>=4.22.0 (from matplotlib)
  Downloading fonttools-4.55.8-cp312-cp312-win_amd64.whl.metadata (103 kB)
Collecting kiwisolver>=1.3.1 (from matplotlib)
  Downloading kiwisolver-1.4.8-cp312-cp312-win_amd64.whl.metadata (6.3 kB)
Requirement already satisfied: numpy>=1.23 in c:\users\smain\appdata\local\progra
ms\python\python312\lib\site-packages (from matplotlib) (2.2.2)
Requirement already satisfied: packaging>=20.0 in c:\users\smain\appdata\roaming
\python\python312\site-packages (from matplotlib) (24.2)
Collecting pillow>=8 (from matplotlib)
  Downloading pillow-11.1.0-cp312-cp312-win_amd64.whl.metadata (9.3 kB)
Collecting pyparsing>=2.3.1 (from matplotlib)
  Using cached pyparsing-3.2.1-py3-none-any.whl.metadata (5.0 kB)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\smain\appdata\roa
ming\python\python312\site-packages (from matplotlib) (2.9.0.post0)
Requirement already satisfied: six>=1.5 in c:\users\smain\appdata\roaming\python
\python312\site-packages (from python-dateutil>=2.7->matplotlib) (1.17.0)
Downloading matplotlib-3.10.0-cp312-cp312-win_amd64.whl (8.0 MB)
   ---------------------------------------- 0.0/8.0 MB ? eta -:--:--
   --- ------------------------------------ 0.8/8.0 MB 3.7 MB/s eta 0:00:02
   ----- ---------------------------------- 1.0/8.0 MB 3.1 MB/s eta 0:00:03
   --------- ------------------------------ 1.8/8.0 MB 2.9 MB/s eta 0:00:03
   ---------- ----------------------------- 2.1/8.0 MB 3.0 MB/s eta 0:00:02
   ----------- ---------------------------- 2.4/8.0 MB 2.5 MB/s eta 0:00:03
   ----------- ---------------------------- 2.4/8.0 MB 2.5 MB/s eta 0:00:03
   ----------- ---------------------------- 2.4/8.0 MB 2.5 MB/s eta 0:00:03
   ----------- ---------------------------- 2.4/8.0 MB 2.5 MB/s eta 0:00:03
   ----------- ---------------------------- 2.4/8.0 MB 2.5 MB/s eta 0:00:03
   ----------- ---------------------------- 2.4/8.0 MB 2.5 MB/s eta 0:00:03
   ----------- ---------------------------- 2.4/8.0 MB 2.5 MB/s eta 0:00:03
   ----------- ---------------------------- 2.4/8.0 MB 2.5 MB/s eta 0:00:03
   ----------- ---------------------------- 2.4/8.0 MB 2.5 MB/s eta 0:00:03
   ----------- ---------------------------- 2.4/8.0 MB 2.5 MB/s eta 0:00:03
   ----------- ---------------------------- 2.4/8.0 MB 2.5 MB/s eta 0:00:03
   ------------- -------------------------- 2.6/8.0 MB 743.8 kB/s eta 0:00:08
   ------------- -------------------------- 2.6/8.0 MB 743.8 kB/s eta 0:00:08
   ------------- -------------------------- 2.6/8.0 MB 743.8 kB/s eta 0:00:08
   ------------- -------------------------- 2.6/8.0 MB 743.8 kB/s eta 0:00:08
   -------------- ------------------------- 2.9/8.0 MB 660.5 kB/s eta 0:00:08
   -------------- ------------------------- 2.9/8.0 MB 660.5 kB/s eta 0:00:08
   --------------- ------------------------ 3.1/8.0 MB 663.8 kB/s eta 0:00:08
   ---------------- ----------------------- 3.4/8.0 MB 684.7 kB/s eta 0:00:07
   ------------------ --------------------- 3.7/8.0 MB 708.1 kB/s eta 0:00:07
   ------------------ --------------------- 3.7/8.0 MB 708.1 kB/s eta 0:00:07
   ------------------- -------------------- 3.9/8.0 MB 722.7 kB/s eta 0:00:06
   -------------------- ------------------- 4.2/8.0 MB 727.4 kB/s eta 0:00:06
   ---------------------- ----------------- 4.5/8.0 MB 747.8 kB/s eta 0:00:05
   ---------------------- ----------------- 4.5/8.0 MB 747.8 kB/s eta 0:00:05
   ----------------------- ---------------- 4.7/8.0 MB 738.9 kB/s eta 0:00:05
   ------------------------ --------------- 5.0/8.0 MB 766.5 kB/s eta 0:00:04
   ------------------------ --------------- 5.0/8.0 MB 766.5 kB/s eta 0:00:04
   ------------------------ --------------- 5.0/8.0 MB 766.5 kB/s eta 0:00:04
```

```
   ------------------------- ------------- 5.2/8.0 MB 729.5 kB/s eta 0:00:04
   -------------------------- ------------ 5.5/8.0 MB 747.3 kB/s eta 0:00:04
   --------------------------- ----------- 5.8/8.0 MB 754.4 kB/s eta 0:00:04
   ------------------------------ -------- 6.3/8.0 MB 798.9 kB/s eta 0:00:03
   ------------------------------- ------- 6.6/8.0 MB 823.4 kB/s eta 0:00:02
   -------------------------------- ------ 6.8/8.0 MB 825.6 kB/s eta 0:00:02
   ----------------------------------- --- 7.3/8.0 MB 874.5 kB/s eta 0:00:01
   ---------------------------------------- 8.0/8.0 MB 930.4 kB/s eta 0:00:00
Downloading contourpy-1.3.1-cp312-cp312-win_amd64.whl (220 kB)
Using cached cycler-0.12.1-py3-none-any.whl (8.3 kB)
Downloading fonttools-4.55.8-cp312-cp312-win_amd64.whl (2.2 MB)
   ---------------------------------------- 0.0/2.2 MB ? eta -:--:--
   --------- ------------------------------ 0.5/2.2 MB 3.4 MB/s eta 0:00:01
   ------------------- -------------------- 1.0/2.2 MB 3.1 MB/s eta 0:00:01
   -------------------------------- ------ 1.8/2.2 MB 3.2 MB/s eta 0:00:01
   ---------------------------------------- 2.2/2.2 MB 3.2 MB/s eta 0:00:00
Downloading kiwisolver-1.4.8-cp312-cp312-win_amd64.whl (71 kB)
Downloading pillow-11.1.0-cp312-cp312-win_amd64.whl (2.6 MB)
   ---------------------------------------- 0.0/2.6 MB ? eta -:--:--
   --- ------------------------------------ 0.3/2.6 MB ? eta -:--:--
   ----------- ---------------------------- 0.8/2.6 MB 2.2 MB/s eta 0:00:01
   --------------- ------------------------ 1.0/2.6 MB 1.9 MB/s eta 0:00:01
   ------------------- -------------------- 1.3/2.6 MB 1.6 MB/s eta 0:00:01
   ----------------------- ---------------- 1.6/2.6 MB 1.8 MB/s eta 0:00:01
   ------------------------------------ ---- 2.4/2.6 MB 2.0 MB/s eta 0:00:01
   ---------------------------------------- 2.6/2.6 MB 2.1 MB/s eta 0:00:00
Using cached pyparsing-3.2.1-py3-none-any.whl (107 kB)
Installing collected packages: pyparsing, pillow, kiwisolver, fonttools, cycler,
contourpy, matplotlib
Successfully installed contourpy-1.3.1 cycler-0.12.1 fonttools-4.55.8 kiwisolver-
1.4.8 matplotlib-3.10.0 pillow-11.1.0 pyparsing-3.2.1
Note: you may need to restart the kernel to use updated packages.
```

```
  WARNING: The scripts fonttools.exe, pyftmerge.exe, pyftsubset.exe and ttx.exe a
re installed in 'c:\Users\smain\AppData\Local\Programs\Python\Python312\Scripts'
which is not on PATH.
  Consider adding this directory to PATH or, if you prefer to suppress this warni
ng, use --no-warn-script-location.
```

In [2]:

```
%pip install numpy==1.26.4
```

```
Collecting numpy==1.26.4
  Downloading numpy-1.26.4-cp312-cp312-win_amd64.whl.metadata (61 kB)
Downloading numpy-1.26.4-cp312-cp312-win_amd64.whl (15.5 MB)
   ---------------------------------------- 0.0/15.5 MB ? eta -:--:--
   ---------------- ----------------------- 6.6/15.5 MB 50.3 MB/s eta 0:00:01
   ------------------- -------------------- 7.6/15.5 MB 18.8 MB/s eta 0:00:01
   ----------------------- ---------------- 9.2/15.5 MB 15.9 MB/s eta 0:00:01
   -------------------------- ------------- 10.2/15.5 MB 12.0 MB/s eta 0:00:01
   ------------------------------ --------- 11.8/15.5 MB 11.5 MB/s eta 0:00:01
   ---------------------------------- ----- 13.4/15.5 MB 10.5 MB/s eta 0:00:01
   ------------------------------------- -- 14.7/15.5 MB 9.8 MB/s eta 0:00:01
   ---------------------------------------  15.2/15.5 MB 8.9 MB/s eta 0:00:01
   ---------------------------------------- 15.5/15.5 MB 8.8 MB/s eta 0:00:00
Installing collected packages: numpy
  Attempting uninstall: numpy
    Found existing installation: numpy 2.2.2
    Uninstalling numpy-2.2.2:
      Successfully uninstalled numpy-2.2.2
Successfully installed numpy-1.26.4
Note: you may need to restart the kernel to use updated packages.
```

```
  WARNING: The script f2py.exe is installed in 'c:\Users\smain\AppData\Local\Programs\Python\Python312\Scripts' which is not on PATH.
  Consider adding this directory to PATH or, if you prefer to suppress this warning, use --no-warn-script-location.
  WARNING: Failed to remove contents in a temporary directory 'C:\Users\smain\AppData\Local\Programs\Python\Python312\Lib\site-packages\~umpy.libs'.
  You can safely remove it manually.
  WARNING: Failed to remove contents in a temporary directory 'C:\Users\smain\AppData\Local\Programs\Python\Python312\Lib\site-packages\~umpy'.
  You can safely remove it manually.
```

In [5]:

```
%pip install scikit-learn
```

```
Collecting scikit-learn
  Downloading scikit_learn-1.6.1-cp312-cp312-win_amd64.whl.metadata (15 kB)
Requirement already satisfied: numpy>=1.19.5 in c:\users\smain\appdata\local\prog
rams\python\python312\lib\site-packages (from scikit-learn) (1.26.4)
Collecting scipy>=1.6.0 (from scikit-learn)
  Downloading scipy-1.15.1-cp312-cp312-win_amd64.whl.metadata (60 kB)
Collecting joblib>=1.2.0 (from scikit-learn)
  Downloading joblib-1.4.2-py3-none-any.whl.metadata (5.4 kB)
Collecting threadpoolctl>=3.1.0 (from scikit-learn)
  Downloading threadpoolctl-3.5.0-py3-none-any.whl.metadata (13 kB)
Downloading scikit_learn-1.6.1-cp312-cp312-win_amd64.whl (11.1 MB)
   ---------------------------------------- 0.0/11.1 MB ? eta -:--:--
   ----- ---------------------------------- 1.6/11.1 MB 12.0 MB/s eta 0:00:01
   ---------- ----------------------------- 2.9/11.1 MB 7.0 MB/s eta 0:00:02
   ------------ --------------------------- 3.4/11.1 MB 5.8 MB/s eta 0:00:02
   ----------------- ---------------------- 5.0/11.1 MB 5.9 MB/s eta 0:00:02
   -------------------- ------------------- 5.8/11.1 MB 5.4 MB/s eta 0:00:01
   ---------------------- ----------------- 6.3/11.1 MB 5.1 MB/s eta 0:00:01
   ------------------------- -------------- 7.1/11.1 MB 4.8 MB/s eta 0:00:01
   --------------------------- ------------ 7.6/11.1 MB 4.6 MB/s eta 0:00:01
   ------------------------------ --------- 8.4/11.1 MB 4.4 MB/s eta 0:00:01
   -------------------------------- ------- 8.9/11.1 MB 4.3 MB/s eta 0:00:01
   ---------------------------------- ----- 9.7/11.1 MB 4.2 MB/s eta 0:00:01
   ------------------------------------ --- 10.2/11.1 MB 4.1 MB/s eta 0:00:01
   ---------------------------------------  11.0/11.1 MB 4.0 MB/s eta 0:00:01
   ---------------------------------------- 11.1/11.1 MB 3.9 MB/s eta 0:00:00
Downloading joblib-1.4.2-py3-none-any.whl (301 kB)
Downloading scipy-1.15.1-cp312-cp312-win_amd64.whl (43.6 MB)
   ---------------------------------------- 0.0/43.6 MB ? eta -:--:--
   ---------------------------------------- 0.5/43.6 MB 2.8 MB/s eta 0:00:16
   - -------------------------------------- 1.3/43.6 MB 3.0 MB/s eta 0:00:14
   - -------------------------------------- 1.8/43.6 MB 3.1 MB/s eta 0:00:14
   -- ------------------------------------- 2.6/43.6 MB 3.1 MB/s eta 0:00:14
   -- ------------------------------------- 3.1/43.6 MB 3.1 MB/s eta 0:00:13
   --- ------------------------------------ 3.9/43.6 MB 3.1 MB/s eta 0:00:13
   ---- ----------------------------------- 4.5/43.6 MB 3.1 MB/s eta 0:00:13
   ---- ----------------------------------- 5.2/43.6 MB 3.1 MB/s eta 0:00:13
   ----- ---------------------------------- 5.8/43.6 MB 3.1 MB/s eta 0:00:13
   ----- ---------------------------------- 6.3/43.6 MB 3.1 MB/s eta 0:00:12
   ------ --------------------------------- 7.1/43.6 MB 3.1 MB/s eta 0:00:12
   ------ --------------------------------- 7.6/43.6 MB 3.1 MB/s eta 0:00:12
   ------- -------------------------------- 8.4/43.6 MB 3.1 MB/s eta 0:00:12
   -------- ------------------------------- 8.9/43.6 MB 3.1 MB/s eta 0:00:12
   -------- ------------------------------- 9.7/43.6 MB 3.1 MB/s eta 0:00:11
   --------- ------------------------------ 10.5/43.6 MB 3.1 MB/s eta 0:00:11
   ---------- ----------------------------- 11.0/43.6 MB 3.1 MB/s eta 0:00:11
   ---------- ----------------------------- 11.5/43.6 MB 3.1 MB/s eta 0:00:11
   ----------- ---------------------------- 12.3/43.6 MB 3.1 MB/s eta 0:00:10
   ------------ --------------------------- 13.1/43.6 MB 3.1 MB/s eta 0:00:10
   ------------ --------------------------- 13.9/43.6 MB 3.2 MB/s eta 0:00:10
   ------------- -------------------------- 14.9/43.6 MB 3.3 MB/s eta 0:00:09
   -------------- ------------------------- 15.5/43.6 MB 3.3 MB/s eta 0:00:09
   -------------- ------------------------- 16.3/43.6 MB 3.2 MB/s eta 0:00:09
   --------------- ------------------------ 16.8/43.6 MB 3.3 MB/s eta 0:00:09
   ---------------- ----------------------- 17.6/43.6 MB 3.2 MB/s eta 0:00:09
   ---------------- ----------------------- 18.1/43.6 MB 3.2 MB/s eta 0:00:08
   ----------------- ---------------------- 18.9/43.6 MB 3.2 MB/s eta 0:00:08
   ------------------ --------------------- 19.7/43.6 MB 3.2 MB/s eta 0:00:08
   ------------------ --------------------- 20.7/43.6 MB 3.3 MB/s eta 0:00:07
   ------------------- -------------------- 21.5/43.6 MB 3.3 MB/s eta 0:00:07
```

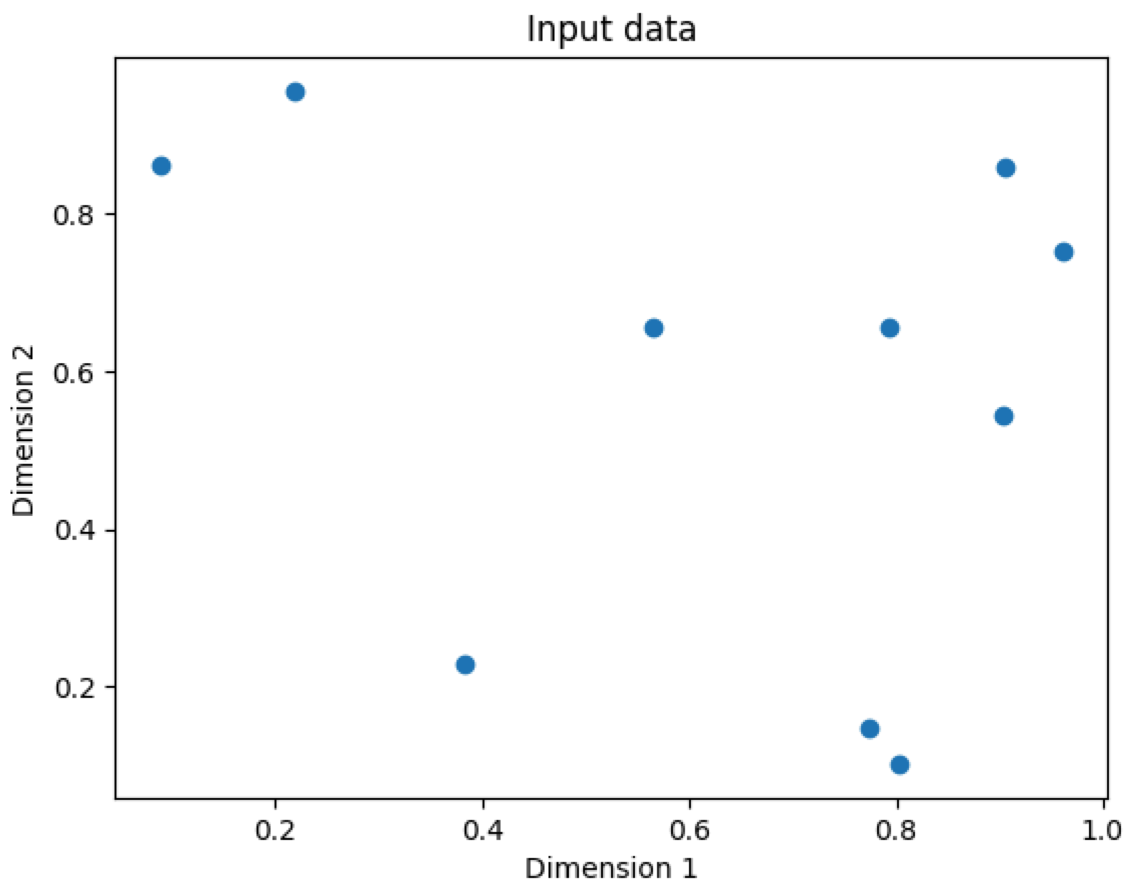
```
   -------------------- ------------------- 22.0/43.6 MB 3.3 MB/s eta 0:00:07
   -------------------- ------------------- 22.8/43.6 MB 3.3 MB/s eta 0:00:07
   --------------------- ------------------ 23.3/43.6 MB 3.3 MB/s eta 0:00:07
   ---------------------- ----------------- 24.1/43.6 MB 3.3 MB/s eta 0:00:06
   ---------------------- ----------------- 24.6/43.6 MB 3.3 MB/s eta 0:00:06
   ----------------------- ---------------- 25.2/43.6 MB 3.3 MB/s eta 0:00:06
   ----------------------- ---------------- 26.0/43.6 MB 3.3 MB/s eta 0:00:06
   ------------------------ --------------- 26.7/43.6 MB 3.3 MB/s eta 0:00:06
   ------------------------ --------------- 27.3/43.6 MB 3.3 MB/s eta 0:00:05
   ------------------------- -------------- 28.0/43.6 MB 3.3 MB/s eta 0:00:05
   -------------------------- ------------- 28.8/43.6 MB 3.3 MB/s eta 0:00:05
   --------------------------- ------------ 29.9/43.6 MB 3.4 MB/s eta 0:00:05
   ---------------------------- ----------- 31.2/43.6 MB 3.4 MB/s eta 0:00:04
   ------------------------------ --------- 32.8/43.6 MB 3.5 MB/s eta 0:00:04
   ------------------------------- -------- 34.1/43.6 MB 3.6 MB/s eta 0:00:03
   -------------------------------- ------- 35.4/43.6 MB 3.6 MB/s eta 0:00:03
   --------------------------------- ------ 37.0/43.6 MB 3.7 MB/s eta 0:00:02
   ----------------------------------- ---- 38.3/43.6 MB 3.7 MB/s eta 0:00:02
   ------------------------------------ --- 39.6/43.6 MB 3.8 MB/s eta 0:00:02
   ------------------------------------- -- 40.9/43.6 MB 3.8 MB/s eta 0:00:01
   -------------------------------------- - 42.2/43.6 MB 3.9 MB/s eta 0:00:01
   ---------------------------------------  43.5/43.6 MB 3.9 MB/s eta 0:00:01
   ---------------------------------------- 43.6/43.6 MB 3.9 MB/s eta 0:00:00
Downloading threadpoolctl-3.5.0-py3-none-any.whl (18 kB)
Installing collected packages: threadpoolctl, scipy, joblib, scikit-learn
Successfully installed joblib-1.4.2 scikit-learn-1.6.1 scipy-1.15.1 threadpoolctl
-3.5.0
Note: you may need to restart the kernel to use updated packages.
```

In [2]:

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

# Générer des données aléatoires
data = np.random.rand(10, 2)  # 10 exemples avec 2 dimensions
labels = np.random.randint(0, 2, size=(10, 1))  # 10 labels binaires (0 ou 1)

# Combiner les données et les labels
text = np.hstack((data, labels))

# Sauvegarder dans un fichier texte
np.savetxt('data_perceptron.txt', text)

print("Le fichier 'data_perceptron.txt' a été généré avec succès.")

# Load input data
text = np.loadtxt('data_perceptron.txt')

# Separate datapoints and labels
data = text[:, :2]
labels = text[:, 2].reshape((text.shape[0], 1))

# Plot input data
plt.figure()
plt.scatter(data[:,0], data[:,1])
plt.xlabel('Dimension 1')
plt.ylabel('Dimension 2')
plt.title('Input data')

# Define minimum and maximum values for each dimension
```

```python
dim1_min, dim1_max, dim2_min, dim2_max = 0, 1, 0, 1

# Number of neurons in the output layer
num_output = labels.shape[1]

# Define a perceptron with 2 input neurons (because we
# have 2 dimensions in the input data)
dim1 = [dim1_min, dim1_max]
dim2 = [dim2_min, dim2_max]
perceptron = nl.net.newp([dim1, dim2], num_output)

# Train the perceptron using the data
error_progress = perceptron.train(data, labels, epochs=100, show=20, lr=0.03)

# Plot the training progress
plt.figure()
plt.plot(error_progress)
plt.xlabel('Number of epochs')
plt.ylabel('Training error')
plt.title('Training error progress')
plt.grid()

plt.show()
```

```
Le fichier 'data_perceptron.txt' a été généré avec succès.
The goal of learning is reached
```



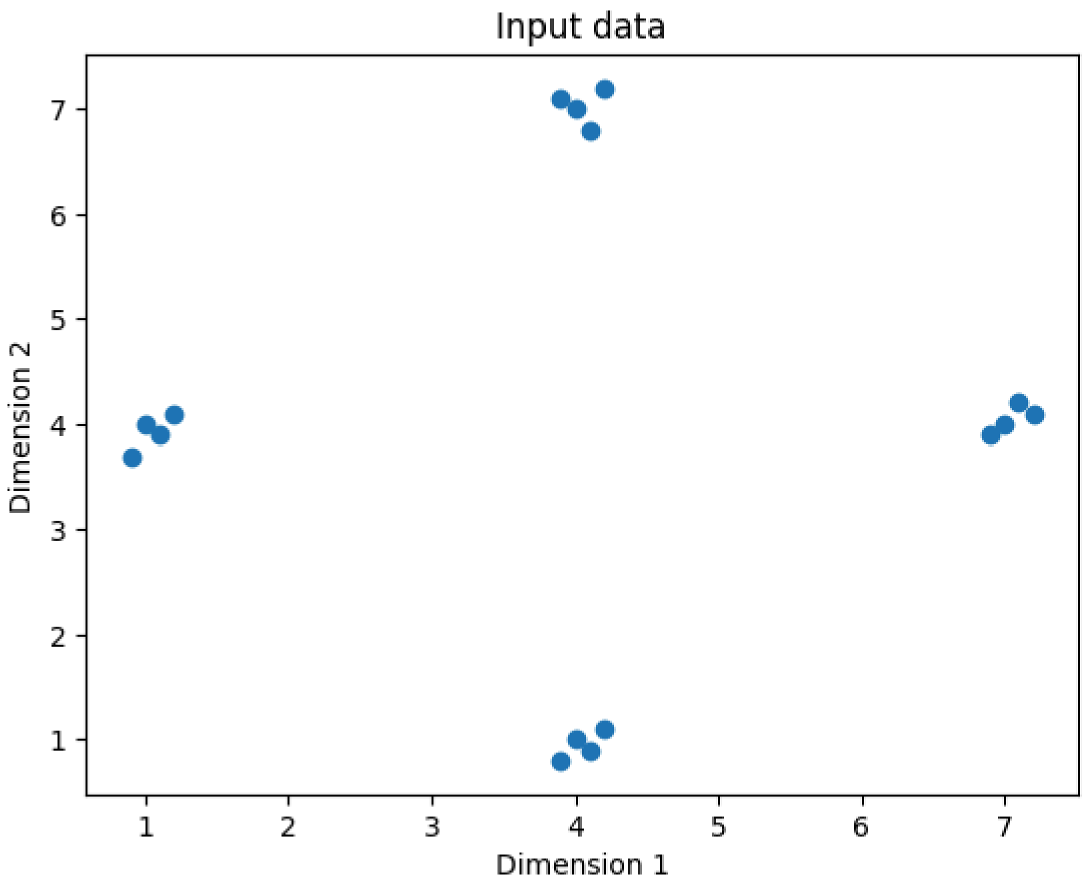
Ce bloc de code ci dessus génère un ensemble de données aléatoires de 10 points avec 2 dimensions et des labels binaires (0 ou 1), puis les sauvegarde dans un fichier texte. Ensuite, il charge ces données et les sépare en variables d'entrée (data) et labels. Il crée un perceptron avec 2 neurones d'entrée et 1 neurone de sortie, puis l'entraîne sur les données pendant 100 époques. Enfin, il affiche un graphique des données d'entrée et un graphique montrant l'évolution de l'erreur pendant l'entraînement.

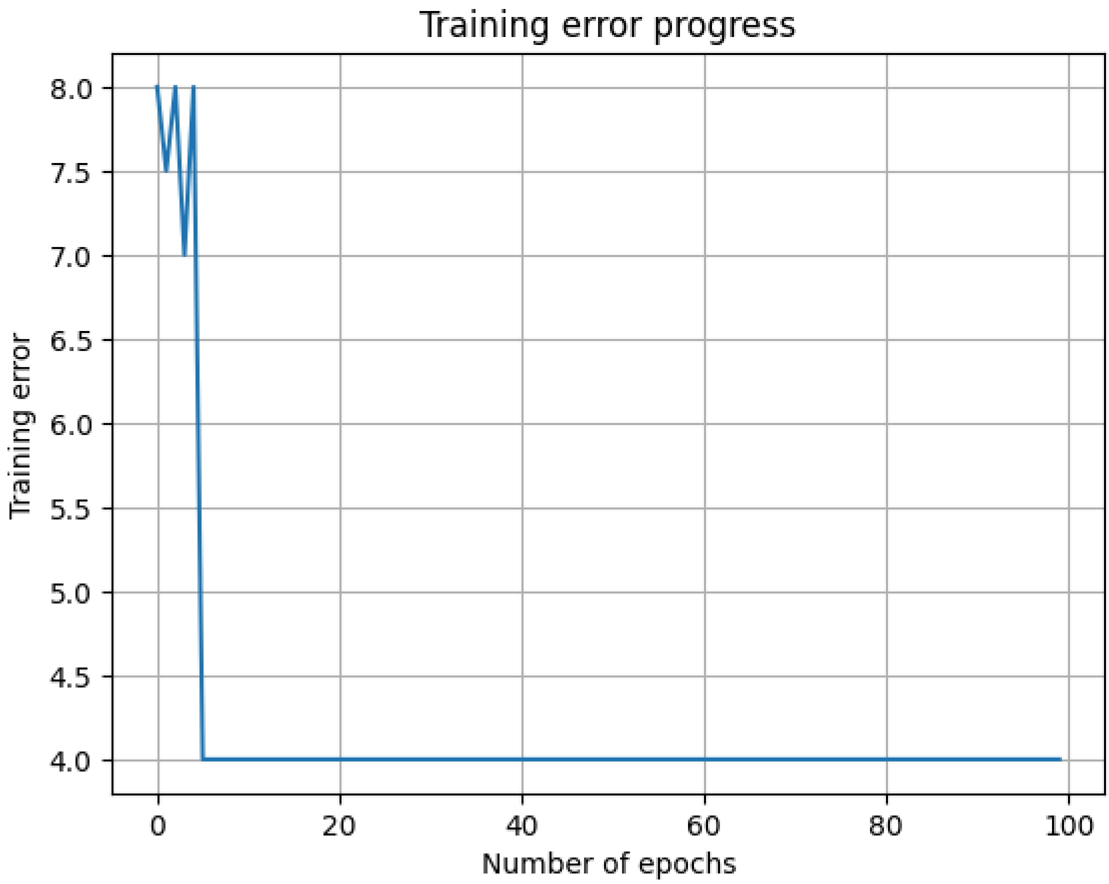
In [2]:

```
import numpy as np
import matplotlib.pyplot as plt
import neurolab as nl

# Load input data
text = np.loadtxt('C:/Users/smain/Downloads/data_simple_nn.txt')

# Separate it into datapoints and labels
data = text[:, 0:2]
labels = text[:, 2:]

# Plot input data
plt.figure()
plt.scatter(data[:,0], data[:,1])
plt.xlabel('Dimension 1')
plt.ylabel('Dimension 2')
plt.title('Input data')

# Minimum and maximum values for each dimension
dim1_min, dim1_max = data[:,0].min(), data[:,0].max()
dim2_min, dim2_max = data[:,1].min(), data[:,1].max()

# Define the number of neurons in the output layer
```

```
num_output = labels.shape[1]

# Define a single-layer neural network
dim1 = [dim1_min, dim1_max]
dim2 = [dim2_min, dim2_max]
nn = nl.net.newp([dim1, dim2], num_output)

# Train the neural network
error_progress = nn.train(data, labels, epochs=100, show=20, lr=0.03)

# Plot the training progress
plt.figure()
plt.plot(error_progress)
plt.xlabel('Number of epochs')
plt.ylabel('Training error')
plt.title('Training error progress')
plt.grid()

plt.show()

# Run the classifier on test datapoints
print('\nTest results:')
data_test = [[0.4, 4.3], [4.4, 0.6], [4.7, 8.1]]
for item in data_test:
    print(item, '-->', nn.sim([item])[0])
```

```
Epoch: 20; Error: 4.0;
Epoch: 40; Error: 4.0;
Epoch: 60; Error: 4.0;
Epoch: 80; Error: 4.0;
Epoch: 100; Error: 4.0;
The maximum number of train epochs is reached
```






```
Test results:
[0.4, 4.3] --> [0. 0.]
[4.4, 0.6] --> [1. 0.]
[4.7, 8.1] --> [1. 1.]
```

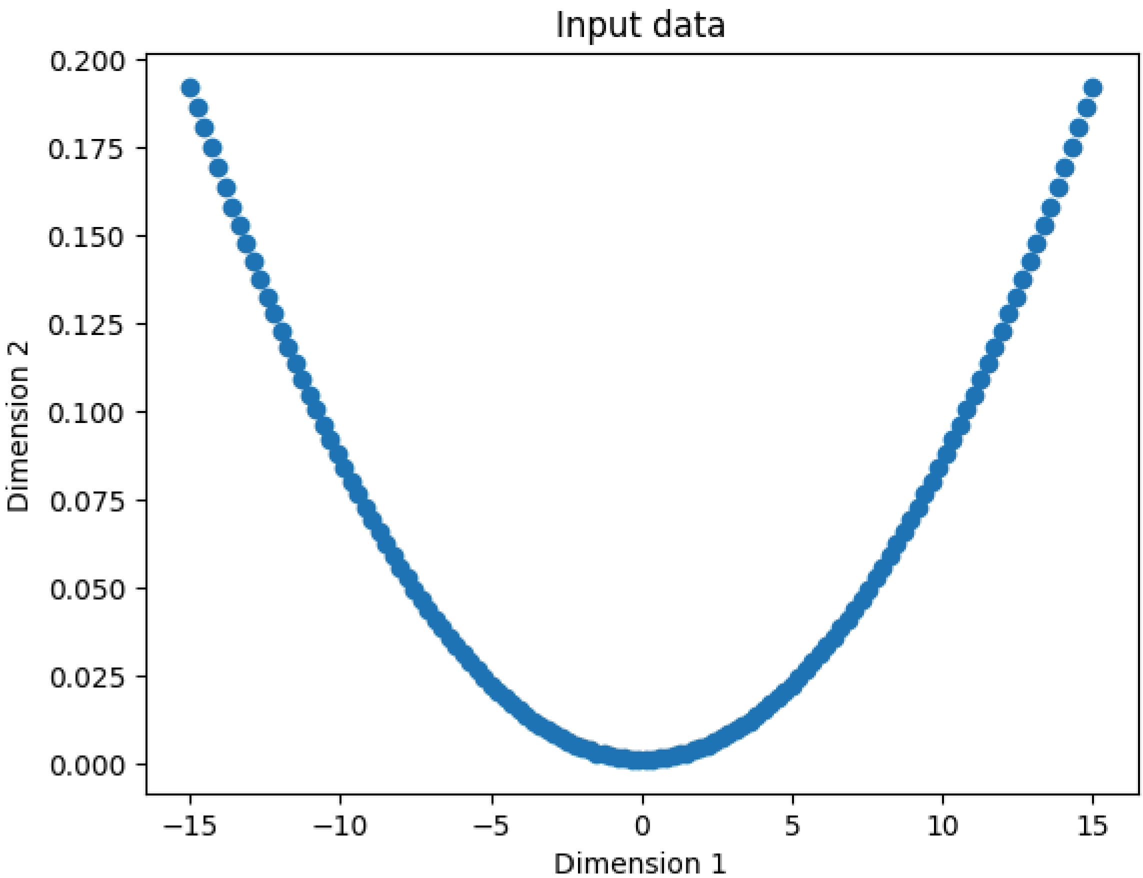
Ce bloc de code ci-dessus charge un fichier de données avec 2 dimensions et des labels associés, puis affiche un graphique des points d'entrée. Il définit un réseau de neurones à couche unique avec 2 neurones d'entrée et entraîne ce réseau sur les données pendant 100 époques. L'évolution de l'erreur d'entraînement est tracée au fil du temps. Après l'entraînement, le réseau est testé sur quelques nouveaux points de données, et les résultats sont affichés pour chaque point testé.

In [3]:

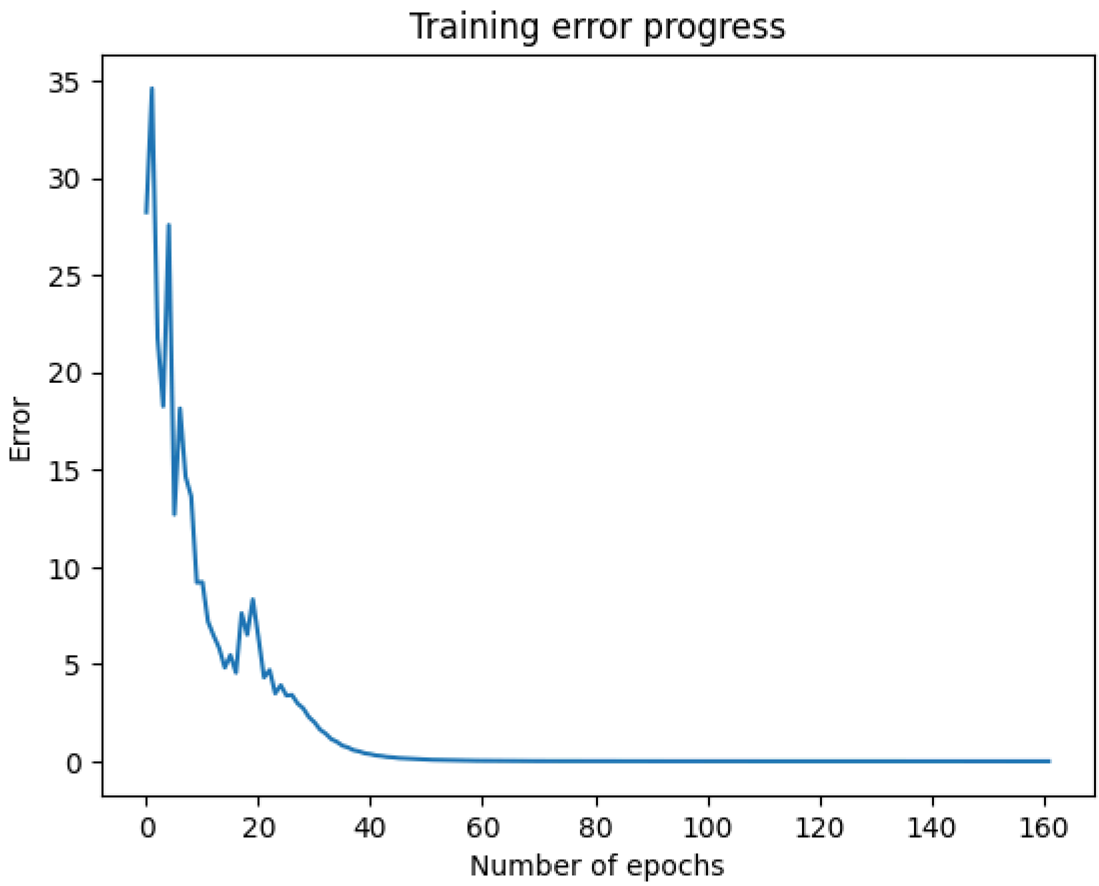
```
import numpy as np
import matplotlib.pyplot as plt
import neurolab as nl

# Generate some training data
min_val = -15
max_val = 15
num_points = 130
x = np.linspace(min_val, max_val, num_points)
y = 3 * np.square(x) + 5
y /= np.linalg.norm(y)

# Create data and labels
data = x.reshape(num_points, 1)
labels = y.reshape(num_points, 1)

# Plot input data
plt.figure()
plt.scatter(data, labels)
```

```
plt.xlabel('Dimension 1')
plt.ylabel('Dimension 2')
plt.title('Input data')

# Define a multilayer neural network with 2 hidden layers;
# First hidden layer consists of 10 neurons
# Second hidden layer consists of 6 neurons
# Output layer consists of 1 neuron
nn = nl.net.newff([[min_val, max_val]], [10, 6, 1])

# Set the training algorithm to gradient descent
nn.trainf = nl.train.train_gd

# Train the neural network
error_progress = nn.train(data, labels, epochs=2000, show=100, goal=0.01)

# Run the neural network on training datapoints
output = nn.sim(data)
y_pred = output.reshape(num_points)

# Plot training error
plt.figure()
plt.plot(error_progress)
plt.xlabel('Number of epochs')
plt.ylabel('Error')
plt.title('Training error progress')

# Plot the output
x_dense = np.linspace(min_val, max_val, num_points * 2)
y_dense_pred = nn.sim(x_dense.reshape(x_dense.size,1)).reshape(x_dense.size)

plt.figure()
plt.plot(x_dense, y_dense_pred, '-', x, y, '.', x, y_pred, 'p')
plt.title('Actual vs predicted')

plt.show()
```

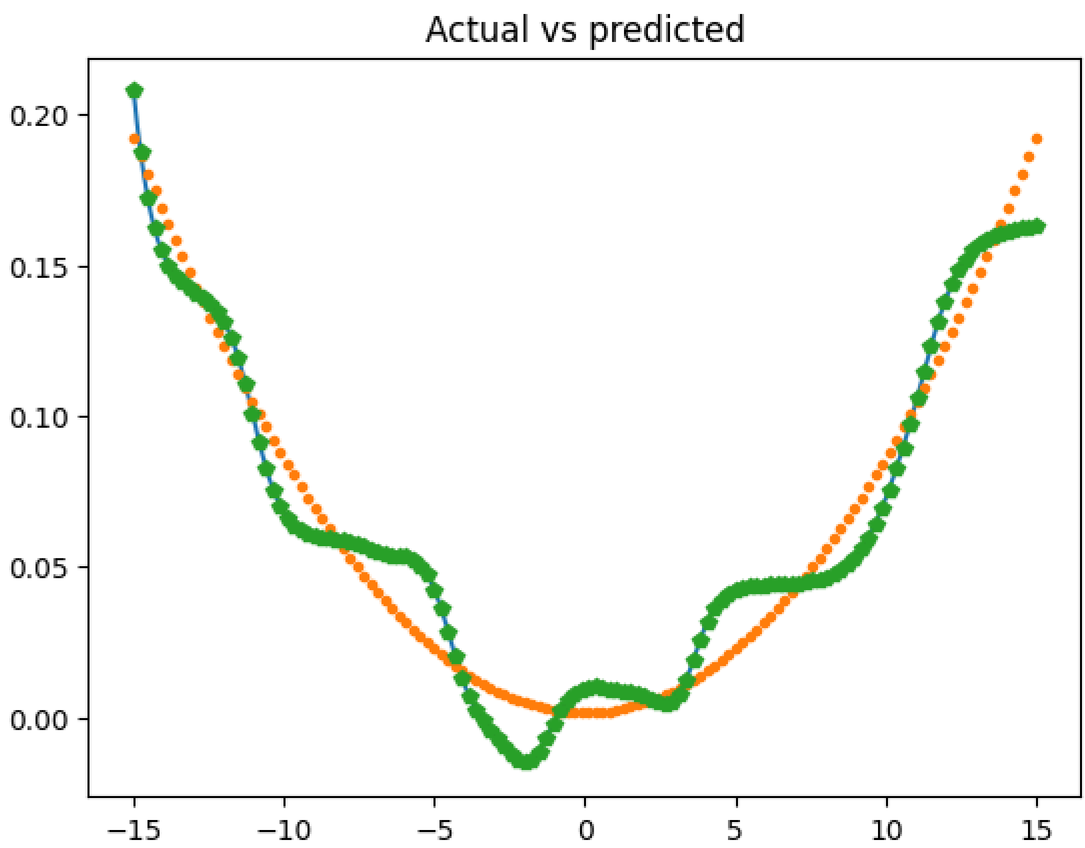
```
Epoch: 100; Error: 0.01281158750101949;
The goal of learning is reached
```



Input data
Dimension 2
Dimension 1
0.200
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
−15
−10
−5
0
5
10
15



Training error progress
Error
Number of epochs
35
30
25
20
15
10
5
0
0
20
40
60
80
100
120
140
160



Ce bloc de code ci-dessus génère des données d'entraînement , puis crée un réseau de neurones multicouche avec 2 couches cachées et une sortie. Le réseau est entraîné à l'aide de la descente de gradient sur plusieurs époques pour prédire les valeurs de y. L'évolution de l'erreur pendant l'entraînement est tracée. Enfin, le réseau prédit les valeurs sur un ensemble de points d'entrée, et un graphique compare les valeurs réelles et prédites.

In [1]:
```
import neurolab.net
print(neurolab.net.__file__)
```

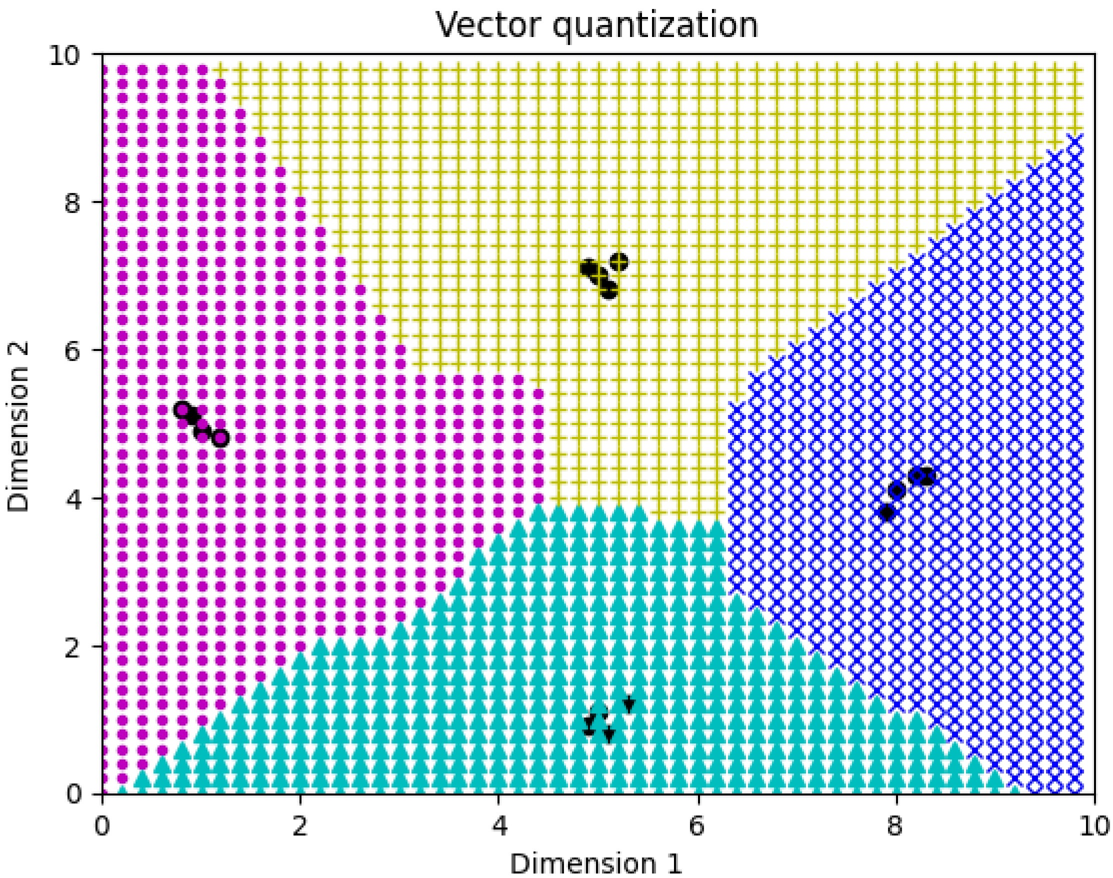
```
c:\Users\smain\AppData\Local\Programs\Python\Python312\Lib\site-packages\neurolab
\net.py
```

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import neurolab as nl
import importlib
import neurolab.net  # Assurez-vous que neurolab est bien importé
importlib.reload(neurolab.net)


# Load input data
text = np.loadtxt('C:/Users/smain/Downloads/data_vector_quantization.txt')

# Separate it into data and labels
data = text[:, 0:2]
labels = text[:, 2:]

# Define a neural network with 2 layers:
# 10 neurons in input layer and 4 neurons in output layer
num_input_neurons = 10
num_output_neurons = 4
```

```python
weights = [1/num_output_neurons] * num_output_neurons
nn = nl.net.newlvq(nl.tool.minmax(data), num_input_neurons, weights)

# Train the neural network
_ = nn.train(data, labels, epochs=500, goal=-1)

# Create the input grid
xx, yy = np.meshgrid(np.arange(0, 10, 0.2), np.arange(0, 10, 0.2))
xx.shape = xx.size, 1
yy.shape = yy.size, 1
grid_xy = np.concatenate((xx, yy), axis=1)

# Evaluate the input grid of points
grid_eval = nn.sim(grid_xy)

# Define the 4 classes
class_1 = data[labels[:,0] == 1]
class_2 = data[labels[:,1] == 1]
class_3 = data[labels[:,2] == 1]
class_4 = data[labels[:,3] == 1]

# Define X-Y grids for all the 4 classes
grid_1 = grid_xy[grid_eval[:,0] == 1]
grid_2 = grid_xy[grid_eval[:,1] == 1]
grid_3 = grid_xy[grid_eval[:,2] == 1]
grid_4 = grid_xy[grid_eval[:,3] == 1]

# Plot the outputs
plt.plot(class_1[:,0], class_1[:,1], 'ko',
        class_2[:,0], class_2[:,1], 'ko',
        class_3[:,0], class_3[:,1], 'ko',
        class_4[:,0], class_4[:,1], 'ko')
plt.plot(grid_1[:,0], grid_1[:,1], 'm.',
        grid_2[:,0], grid_2[:,1], 'bx',
        grid_3[:,0], grid_3[:,1], 'c^',
        grid_4[:,0], grid_4[:,1], 'y+')
plt.axis([0, 10, 0, 10])
plt.xlabel('Dimension 1')
plt.ylabel('Dimension 2')
plt.title('Vector quantization')

plt.show()
```

```
c:\Users\smain\AppData\Local\Programs\Python\Python312\Lib\site-packages\neurolab
\net.py:2: SyntaxWarning: invalid escape sequence '\*'
  """
```

```
Epoch: 100; Error: 0.0;
Epoch: 200; Error: 0.0;
Epoch: 300; Error: 0.0;
Epoch: 400; Error: 0.0;
Epoch: 500; Error: 0.0;
The maximum number of train epochs is reached
```

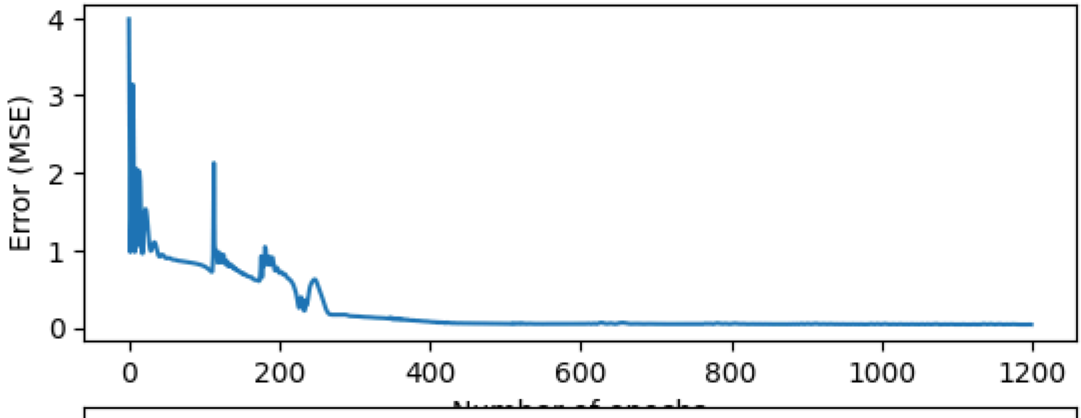


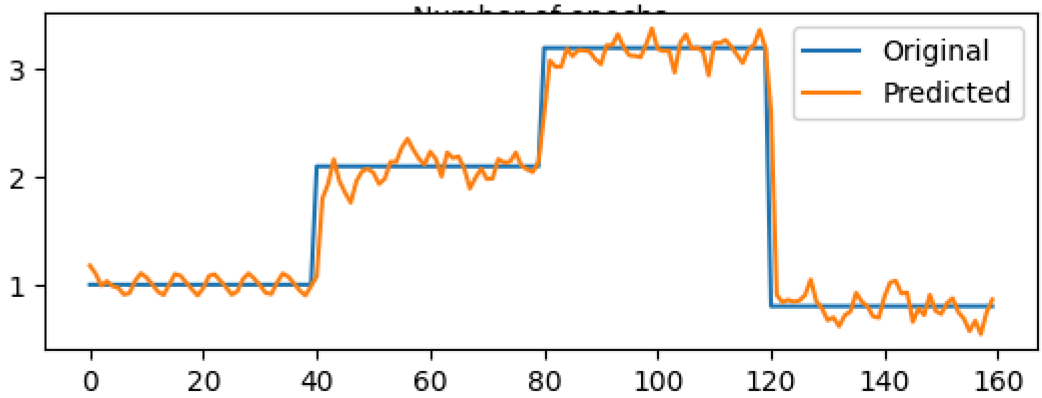
Ce bloc de code ci-dessus charge un jeu de données, le sépare en données et labels, puis crée un réseau de neurones pour la quantification vectorielle avec 10 neurones en entrée et 4 neurones en sortie. Le réseau est entraîné pendant 500 époques pour apprendre à classer les données. Il crée ensuite une grille d'entrée et évalue chaque point de la grille pour attribuer des classes aux zones. Enfin, il affiche un graphique montrant les points de données réels, les classes prédites par le réseau, et les différentes zones de classification.

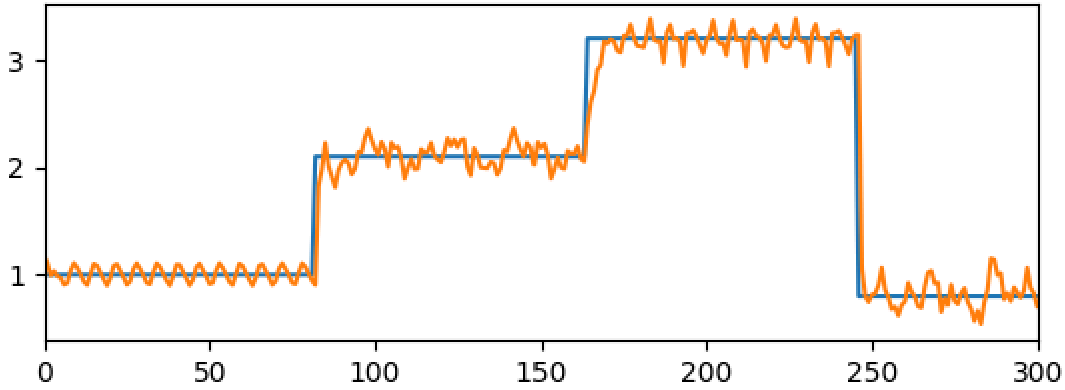
In [2]:
```
import numpy as np
import matplotlib.pyplot as plt
import neurolab as nl

def get_data(num_points):
    # Create sine waveforms
    wave_1 = 0.5 * np.sin(np.arange(0, num_points))
    wave_2 = 3.6 * np.sin(np.arange(0, num_points))
    wave_3 = 1.1 * np.sin(np.arange(0, num_points))
    wave_4 = 4.7 * np.sin(np.arange(0, num_points))

    # Create varying amplitudes
    amp_1 = np.ones(num_points)
    amp_2 = 2.1 + np.zeros(num_points)
    amp_3 = 3.2 * np.ones(num_points)
    amp_4 = 0.8 + np.zeros(num_points)

    wave = np.array([wave_1, wave_2, wave_3, wave_4]).reshape(num_points * 4, 1)
    amp = np.array([[amp_1, amp_2, amp_3, amp_4]]).reshape(num_points * 4, 1)

    return wave, amp

# Visualize the output
```

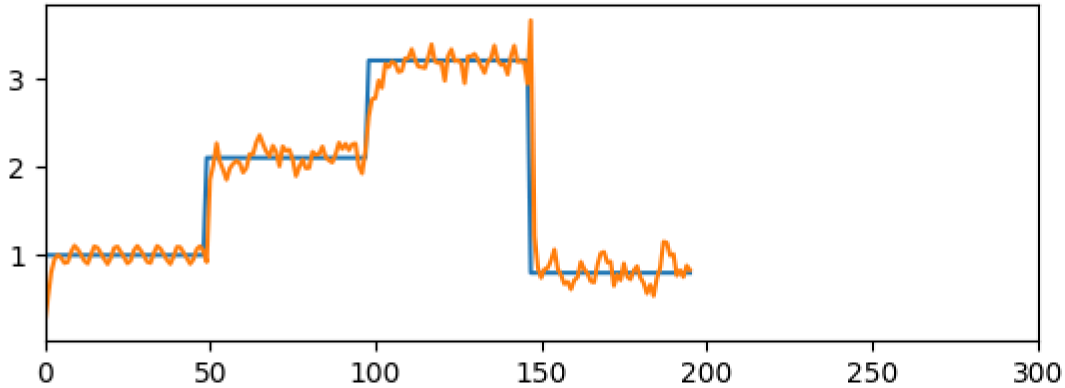
```python
def visualize_output(nn, num_points_test):
    wave, amp = get_data(num_points_test)
    output = nn.sim(wave)
    plt.plot(amp.reshape(num_points_test * 4))
    plt.plot(output.reshape(num_points_test * 4))

if __name__=='__main__':
    # Create some sample data
    num_points = 40
    wave, amp = get_data(num_points)

    # Create a recurrent neural network with 2 layers
    nn = nl.net.newelm([[-2, 2]], [10, 1], [nl.trans.TanSig(), nl.trans.PureLin(

    # Set the init functions for each layer
    nn.layers[0].initf = nl.init.InitRand([-0.1, 0.1], 'wb')
    nn.layers[1].initf = nl.init.InitRand([-0.1, 0.1], 'wb')
    nn.init()

    # Train the recurrent neural network
    error_progress = nn.train(wave, amp, epochs=1200, show=100, goal=0.01)

    # Run the training data through the network
    output = nn.sim(wave)

    # Plot the results
    plt.subplot(211)
    plt.plot(error_progress)
    plt.xlabel('Number of epochs')
    plt.ylabel('Error (MSE)')

    plt.subplot(212)
    plt.plot(amp.reshape(num_points * 4))
    plt.plot(output.reshape(num_points * 4))
    plt.legend(['Original', 'Predicted'])

    # Testing the network performance on unknown data
    plt.figure()

    plt.subplot(211)
    visualize_output(nn, 82)
    plt.xlim([0, 300])

    plt.subplot(212)
    visualize_output(nn, 49)
    plt.xlim([0, 300])

    plt.show()
```

```
Epoch: 100; Error: 0.7993687726843021;
Epoch: 200; Error: 0.7222579588673927;
Epoch: 300; Error: 0.1474527761895535;
Epoch: 400; Error: 0.07834713606921348;
Epoch: 500; Error: 0.05071397408393781;
Epoch: 600; Error: 0.050570220807457864;
Epoch: 700; Error: 0.049063212103352474;
Epoch: 800; Error: 0.0487215184701011;
Epoch: 900; Error: 0.049898340448014125;
Epoch: 1000; Error: 0.04406275933221161;
Epoch: 1100; Error: 0.04329443960369153;
Epoch: 1200; Error: 0.04099483714313155;
The maximum number of train epochs is reached
```









Ce bloc de code ci-dessus génère des données de formes d'ondes sinusoïdales avec différentes amplitudes, puis crée un réseau de neurones récurrent avec 2 couches pour prédire ces amplitudes. Le réseau est entraîné sur 1200 époques pour minimiser l'erreur entre les valeurs originales et les prévisions. Après l'entraînement, les résultats sont visualisés en comparant les valeurs réelles et prédites. Le réseau est également testé sur des données inconnues, et les performances sont affichées sur plusieurs graphiques.

In [3]:
```
import os
import sys

import cv2
import numpy as np

# Define the input file
input_file = "C:/Users/smain/Downloads/letter.data"

# Define the visualization parameters
img_resize_factor = 12
start = 6
end = -1
height, width = 16, 8

# Iterate until the user presses the Esc key
with open(input_file, 'r') as f:
    for line in f.readlines():
        # Read the data
        data = np.array([255 * float(x) for x in line.split('\t')[start:end]])

        # Reshape the data into a 2D image
        img = np.reshape(data, (height, width))

        # Scale the image
        img_scaled = cv2.resize(img, None, fx=img_resize_factor, fy=img_resize_f
```

```
        # Display the image
        cv2.imshow('Image', img_scaled)

        # Check if the user pressed the Esc key
        c = cv2.waitKey()
        if c == 27:
            break
```

```
The Kernel crashed while executing code in the current cell or a previous cell.

Please review the code in the cell(s) to identify a possible cause of the failur
e.

Click <a href='https://aka.ms/vscodeJupyterKernelCrash'>here</a> for more info.

View Jupyter <a href='command:jupyter.viewOutput'>log</a> for further details.
```

Ce bloc de code ci-dessus charge des données à partir d'un fichier texte et les affiche sous forme d'images en utilisant OpenCV. Chaque ligne du fichier représente une image 2D, que le programme redimensionne pour l'afficher dans une fenêtre. L'image est d'abord transformée à partir de valeurs numériques, puis redimensionnée avec un facteur de mise à l'échelle. Le programme continue d'afficher les images jusqu'à ce que l'utilisateur appuie sur la touche Esc, ce qui permet de fermer la fenêtre et d'arrêter l'exécution.

In [2]:

```
import numpy as np
import neurolab as nl

# Define the input file
input_file = "C:/Users/smain/Downloads/letter.data"

# Define the number of datapoints to
# be loaded from the input file
num_datapoints = 50

# String containing all the distinct characters
orig_labels = 'omandig'

# Compute the number of distinct characters
num_orig_labels = len(orig_labels)

# Define the training and testing parameters
num_train = int(0.9 * num_datapoints)
num_test = num_datapoints - num_train

# Define the dataset extraction parameters
start = 6
end = -1

# Creating the dataset
data = []
labels = []
with open(input_file, 'r') as f:
    for line in f.readlines():
        # Split the current line tabwise
        list_vals = line.split('\t')
```

```python
        # Check if the label is in our ground truth
        # labels. If not, we should skip it.
        if list_vals[1] not in orig_labels:
            continue

        # Extract the current label and append it
        # to the main list
        label = np.zeros((num_orig_labels, 1))
        label[orig_labels.index(list_vals[1])] = 1
        labels.append(label)

        # Extract the character vector and append it to the main list
        cur_char = np.array([float(x) for x in list_vals[start:end]])
        data.append(cur_char)

        # Exit the loop once the required dataset has been created
        if len(data) >= num_datapoints:
            break

# Convert the data and labels to numpy arrays
data = np.asfarray(data)
labels = np.array(labels).reshape(num_datapoints, num_orig_labels)

# Extract the number of dimensions
num_dims = len(data[0])

# Create a feedforward neural network
nn = nl.net.newff([[0, 1] for _ in range(len(data[0]))],
        [128, 16, num_orig_labels])

# Set the training algorithm to gradient descent
nn.trainf = nl.train.train_gd

# Train the network
error_progress = nn.train(data[:num_train,:], labels[:num_train,:],
        epochs=10000, show=100, goal=0.01)

# Predict the output for test inputs
print('\nTesting on unknown data:')
predicted_test = nn.sim(data[num_train:, :])
for i in range(num_test):
    print('\nOriginal:', orig_labels[np.argmax(labels[i])])
    print('Predicted:', orig_labels[np.argmax(predicted_test[i])])
```

```
Epoch: 100; Error: 49.86681133330279;
Epoch: 200; Error: 46.53605169767502;
Epoch: 300; Error: 47.174836315891426;
Epoch: 400; Error: 45.96717213446423;
Epoch: 500; Error: 31.479141260661578;
Epoch: 600; Error: 27.518609532000106;
Epoch: 700; Error: 17.754378962834522;
Epoch: 800; Error: 12.524715597241645;
Epoch: 900; Error: 11.664017375190786;
Epoch: 1000; Error: 11.277534358152607;
Epoch: 1100; Error: 14.085984923074596;
Epoch: 1200; Error: 11.854052673373301;
Epoch: 1300; Error: 12.166773582393002;
Epoch: 1400; Error: 11.170014961253429;
Epoch: 1500; Error: 11.50644649866506;
Epoch: 1600; Error: 10.981488625475606;
Epoch: 1700; Error: 10.185264105616955;
Epoch: 1800; Error: 7.66997257161019;
Epoch: 1900; Error: 7.53921194690833;
Epoch: 2000; Error: 6.9480223578908795;
Epoch: 2100; Error: 7.495264582955456;
Epoch: 2200; Error: 7.411151220899443;
Epoch: 2300; Error: 7.401182890076404;
Epoch: 2400; Error: 6.7896690329937925;
Epoch: 2500; Error: 6.652414396346454;
Epoch: 2600; Error: 7.403547570268113;
Epoch: 2700; Error: 6.978748696392439;
Epoch: 2800; Error: 6.68835251382964;
Epoch: 2900; Error: 6.504076263405348;
Epoch: 3000; Error: 6.581941739571798;
Epoch: 3100; Error: 7.42883670754796;
Epoch: 3200; Error: 6.871431046723393;
Epoch: 3300; Error: 6.76905699540862;
Epoch: 3400; Error: 7.430841732432506;
Epoch: 3500; Error: 7.374365177183739;
Epoch: 3600; Error: 7.347425680817782;
Epoch: 3700; Error: 7.276133356755508;
Epoch: 3800; Error: 7.439154762870045;
Epoch: 3900; Error: 6.861709579513123;
Epoch: 4000; Error: 6.826585041469624;
Epoch: 4100; Error: 7.397285855085973;
Epoch: 4200; Error: 7.002521397686387;
Epoch: 4300; Error: 7.262194635143576;
Epoch: 4400; Error: 7.4245263334497755;
Epoch: 4500; Error: 6.723018857611153;
Epoch: 4600; Error: 7.442143665772583;
Epoch: 4700; Error: 7.347850984836979;
Epoch: 4800; Error: 7.120800324436057;
Epoch: 4900; Error: 7.211143875615072;
Epoch: 5000; Error: 7.33240116246006;
Epoch: 5100; Error: 6.622554188048614;
Epoch: 5200; Error: 7.368429528447894;
Epoch: 5300; Error: 7.269349767118068;
Epoch: 5400; Error: 7.22161513596433;
Epoch: 5500; Error: 7.2356890989083436;
Epoch: 5600; Error: 7.208976592063108;
Epoch: 5700; Error: 6.797642851197566;
Epoch: 5800; Error: 7.319113958348281;
Epoch: 5900; Error: 7.174250966259638;
Epoch: 6000; Error: 6.543900669849084;
```

```
Epoch: 6100; Error: 7.025079080450285;
Epoch: 6200; Error: 7.317356124685041;
Epoch: 6300; Error: 7.246922125700408;
Epoch: 6400; Error: 7.147035370403575;
Epoch: 6500; Error: 7.066300470251812;
Epoch: 6600; Error: 7.248962774387925;
Epoch: 6700; Error: 6.6196606504777975;
Epoch: 6800; Error: 7.311186641730032;
Epoch: 6900; Error: 7.248701303927458;
Epoch: 7000; Error: 6.837430034619798;
Epoch: 7100; Error: 7.148087798425853;
Epoch: 7200; Error: 7.039941893573517;
Epoch: 7300; Error: 7.14556376398483;
Epoch: 7400; Error: 6.75818826373548;
Epoch: 7500; Error: 7.248797132729264;
Epoch: 7600; Error: 7.138742707434761;
Epoch: 7700; Error: 6.660953543485601;
Epoch: 7800; Error: 7.089164336577761;
Epoch: 7900; Error: 7.306351622313947;
Epoch: 8000; Error: 6.7739523934209025;
Epoch: 8100; Error: 6.681539775488096;
Epoch: 8200; Error: 7.2225713208550735;
Epoch: 8300; Error: 7.1400359150341;
Epoch: 8400; Error: 6.930898667709393;
Epoch: 8500; Error: 7.169538960326669;
Epoch: 8600; Error: 6.463160123713658;
Epoch: 8700; Error: 7.173060836945318;
Epoch: 8800; Error: 7.041420278795423;
Epoch: 8900; Error: 6.960076590994731;
Epoch: 9000; Error: 7.153834879521806;
Epoch: 9100; Error: 6.6074188548081825;
Epoch: 9200; Error: 7.117930611254483;
Epoch: 9300; Error: 7.00901454296857;
Epoch: 9400; Error: 6.733316271321306;
Epoch: 9500; Error: 7.015895522208584;
Epoch: 9600; Error: 6.8487046280953345;
Epoch: 9700; Error: 5.549084880296286;
Epoch: 9800; Error: 3.8907640462382247;
Epoch: 9900; Error: 0.7664800449155514;
Epoch: 10000; Error: 0.7008511631361978;
The maximum number of train epochs is reached

Testing on unknown data:

Original: o
Predicted: o

Original: m
Predicted: m

Original: m
Predicted: m

Original: a
Predicted: o

Original: n
Predicted: n
```

Ce bloc de code ci-dessus charge un ensemble de données contenant des images de lettres et les associe à des labels correspondants. Il crée un réseau de neurones feedforward avec deux couches cachées pour classer ces lettres en utilisant un algorithme de descente de gradient. Le réseau est entraîné sur 90 % des données et testé sur le reste. Les performances du modèle sont évaluées en comparant les prédictions du réseau aux labels réels des données de test. Le modèle utilise un ensemble de 7 lettres distinctes comme labels.