Exercises for the course
Machine Learning for Data Science
Winter Semester 2024/25

G. Montavon
Institute of Computer Science
Department of Mathematics and Computer Science
Freie Universität Berlin

Exercise Sheet 2 (programming part)

We first load a small handwritten digits dataset provided as part of the scikit-learn library and a few relevant libraries.

In [1]:
```
import matplotlib
%matplotlib inline
from matplotlib import pyplot as plt
import numpy
import torch,torch.optim
import scipy,scipy.spatial
import sklearn,sklearn.datasets

dataset = sklearn.datasets.load_digits()
X = dataset['data']
T = dataset['target']
R = numpy.random.mtrand.RandomState(0).permutation(len(X))[:750]
X = X[R]
T = T[R]
```

Exercise 3: Implementing Metric MDS (15+15+10 P)

In this exercise, we will implement a low-dimensional embedding technique, multi-dimensional scalling (MDS), apply it to the simple handwritten digits dataset above, and visualize the embedded data using a series of scatter plots.

The MDS algorithm does not operate with the dataset directly, but instead, with a matrix of distances. Also, in order to be rather independent from the scale and dimensionality of the data, we would like to normalize the input distance such that distances are on average 10. Lastly, the distance should be provided as a torch tensor so that it can be used for the gradient-based training procedure.

(a) Write a function that takes the data as input and returns a distance matrix according to the specifications above.

In [2]:
```
# ------------------------------
# TODO: replace by your code
# ------------------------------
import solutions
D = solutions.getDistanceMatrix(X)
# ------------------------------


print(f"shape: {str(D.shape)}")
print(f"min:   {D.min():6.3f}")
print(f"mean:  {D.mean():6.3f}")
print(f"max:   {D.max():6.3f}")
```

```
shape: torch.Size([750, 750])
min:    0.000
mean:  10.000
max:   15.391
```

We now would like learn Metric MDS. We will consider a simple gradient-based implementation. The outline of the gradient-based procedure is provided in the code below. The only part missing is the computation of the objective $J(Y)$ where $Y$ are the matrix of size $N \times 2$ containing the coordinates of data points in embedded space. As a reminder, the metric MDS objective the is given by: $$ J(\boldsymbol{y}_1,\dots,\boldsymbol{y}_N) = \left(\frac{\sum_{i<j} \big(d_{ij} - \|\boldsymbol{y}_i - \boldsymbol{y}_j\|\big)^2}{\sum_{i<j} d_{ij}^2}\right)^{1/2} $$

(b) Complete the code below by expressing `J` as a function of `Y` and `D`.

(Hint: Running the training procedure may take up to 5 minutes. For debugging purposes, you can temporarily reduce the number of iterations.)

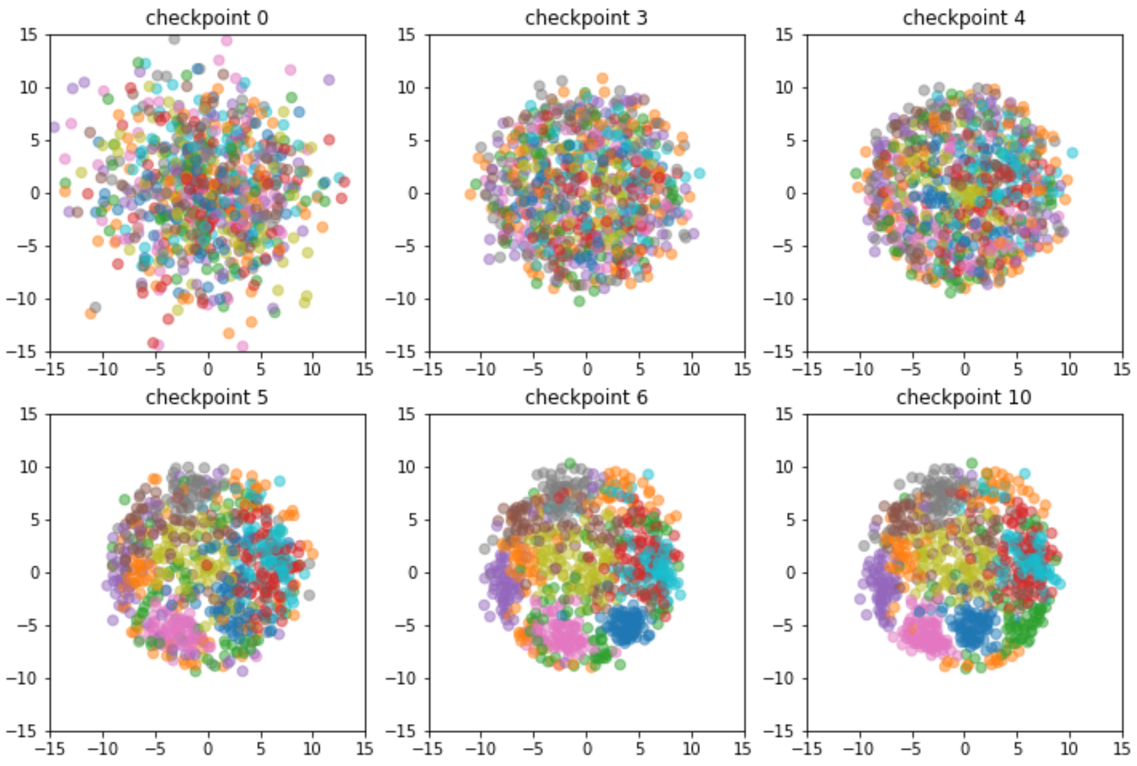
In [3]:
```python
Y = torch.randn([len(X),2])*5
Y.requires_grad_(True)

optimizer = torch.optim.SGD((Y,), lr=100.0, momentum=0.95)

Yhist = []

for i in range(10**4+1):
    optimizer.zero_grad()

    # ------------------------------
    # TODO: replace by your code
    # ------------------------------
    import solutions
    J = solutions.getObjective(Y,D)
    # ------------------------------

    J.backward()

    # Keep track of the objective
    if i in numpy.arange(11)**4:
        print(f"{i:5d} {J.data.numpy():.3f}")
        Yhist += [Y.data.numpy()*1.0]

    optimizer.step()
```

```
    0 0.494
    1 0.494
   16 0.461
   81 0.423
  256 0.409
  625 0.360
 1296 0.338
 2401 0.337
 4096 0.336
 6561 0.335
10000 0.334
```

The code above has stored the embedding at 11 distinct checkpoints (available in `Yhist`). This allows to visualize how the embedding (initialized at random) evolves throughout training, and how it progressively progressively improves the representation of distances in the data.

(c) Using matplotlib, plot the embedding at checkpoints {0,3,4,5,6,10}, and color-code the embedding according to the digit's class *(Hint: because there are 10 classes, you can use of the colormap `tab10` in matplotlib.)*

In [4]:
```python
# ------------------------------
# TODO: replace by your code
# ------------------------------
import solutions
solutions.plotEmbeddings(Yhist,T)
# ------------------------------
```



We observe that during training, data points travel in embedded space. In the final configuration, embedded data forms clusters of points of same class.