Deep Learning (IST, 2021-22)

Practical 7: PCA and Autoencoders

Gonçalo Faria, André Martins, Andreas Wichert, Luis Sá-Couto

Question 1

Consider the following training data:

$$\boldsymbol{x}^{(1)} = \begin{bmatrix} 2.5 \\ 2.4 \end{bmatrix}, \quad \boldsymbol{x}^{(2)} = \begin{bmatrix} 0.5 \\ 0.7 \end{bmatrix}, \quad \boldsymbol{x}^{(3)} = \begin{bmatrix} 2.2 \\ 2.9 \end{bmatrix}, \quad \boldsymbol{x}^{(4)} = \begin{bmatrix} 1.9 \\ 2.2 \end{bmatrix}$$

1. Find the set of orthogonal directions that maximize the variance of the training data. (Hint: perform principal component analysis.)
2. Draw a Cartesian plane containing the training data in the original coordinates and the vectors of principal components.
3. What is the first principal component?
4. Reduce the dimensionality of the training data by mapping the points onto the principal component
5. Find the orthogonal projection of the data onto the first principal component's subspace in the original coordinates.
6. Draw the projected training data on the previous Cartesian plane.
7. What is the mean squared error of the projected training data?

Question 2

Now it's time to run PCA on real data.

1. Load the UCI handwritten digits dataset using `scikit-learn`:

```
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split

from sklearn.datasets import load_digits
data = load_digits()

X, y = data.data, data.target
# normalize images.
X = MinMaxScaler().fit_transform(X)

noise = rng.normal(scale=0.25, size=X_train.shape)
X_train_noisy = X_train + noise
```

This is a dataset containing 1797 8x8 input images of digits, each corresponding to one out of 10 output classes. You can print the dataset description and visualize some input examples with:

```
def plot_digits(X, title):
    """Small helper function to plot 100 digits."""
    fig, axs = plt.subplots(nrows=10, ncols=10, figsize=(8, 8))
    for img, ax in zip(X, axs.ravel()):
        ax.imshow(img.reshape((8, 8)), cmap="Greys")
        ax.axis("off")
    fig.suptitle(title, fontsize=24)
```

Randomly split this data into training (80%) and test (20%) partitions. This can be done with:

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, stratify=y, random_state=0, train_size=1697, test_size=100
)
```

2. Apply independent Gaussian noise to each of the pixels $\epsilon \sim \mathcal{N}(0, 0.25)$ of every image. Plot the resulting images using `matplotlib`.

```
train_noise = 0.25 * np.random.randn(*X_train.shape)
```

3. Run your implementation of PCA on this dataset. Consider different numbers of principal components, and calculate the corresponding orthogonal projection of the images onto the first principal component's subspace in the original coordinate space. Can you recover the uncorrupted digits?

Question 3

Now it's time to train an autoencoder, as depicted in Figure 1, on real data. We will use the same data as the previous exercise.

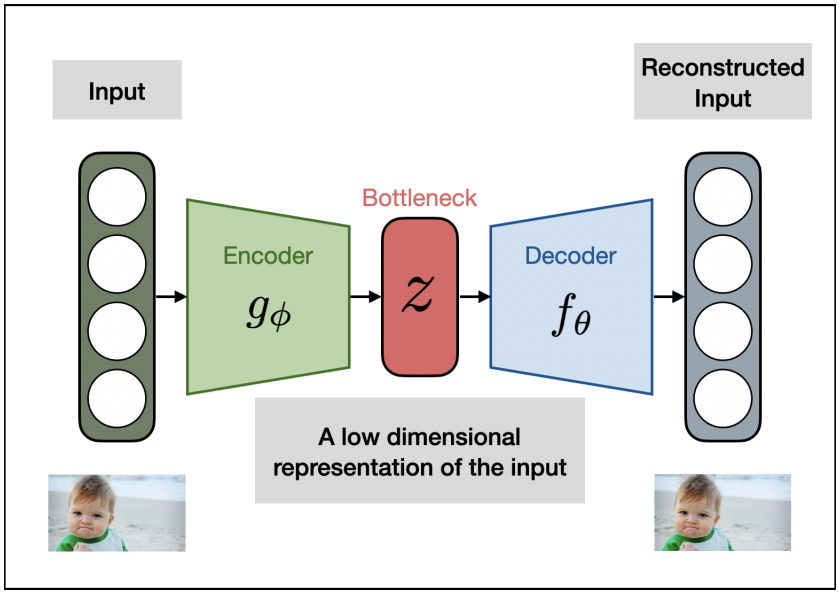

Figura 1: Diagram of an autoencoder.

1. Run your implementation of autoencoders with Linear activations on this dataset. Consider different optimizers and hyper-parameters (particularly bottleneck size).

   Can you see similarities with PCA?

2. Run your implementation of autoencoders with non-linear activations on this dataset. Consider different optimizers and hyper-parameters (particularly bottleneck size). Plot the reconstructed digits on the noisy test set.

3. Rerun your implementation of autoencoders with non-linear activations on this dataset with a bottleneck size of 3. Plot for each test set image its 3-dimensional representation using your model, and color each point according to the image's class.