Artificial neural networks - Exercise session 3

Deep feature learning

Sonny Achten, Bram De Cooman, Hannes De Meulemeester and David Winant

2022-2023

1 Principal Component Analysis

1.1 Introduction

Principal Component Analysis (PCA) involves projecting onto the eigenvectors of the covariance matrix. The basic idea behind PCA is to map a vector $x = (x_1, x_2, \ldots, x_p)$ of a p dimensional space to a lower-dimensional vector $z = (z_1, z_2, \ldots, z_q)$ in a q dimensional space (where $q < p$). We are going to only consider linear mappings, these are ones where $z_d = e_d^T x$ for some unknown column vectors e_d and T denotes vector (and later on, matrix) transposition. In other words, z can be obtained from x by simple matrix multiplication:

$$z = E^T x \tag{1}$$

where E is a p by q matrix whose column are e_d: $E = [e_1, e_2, \ldots, e_q]$. Our goal is then to reconstruct as well as possible the original vectors by using another matrix F such that

$$\hat{x} = Fz = FE^T x \tag{2}$$

where F is a p by q matrix, and $\hat{x}$ resembles x as good as possible. The idea is that any low dimensional intermediate representation z which allows the original data x to be well reconstructed should have captured a lot of the important structure of the data and so will be interesting or useful to work with.

PCA projects the data onto the subspace spanned by the q eigenvectors corresponding to the q largest eigenvalues of the correlation matrix. More concrete, in order to perform a PCA reduction of a dataset containing N datapoints of dimension p, one can use the following algorithm:

- Zero-mean the data by subtracting the mean of the dataset from each datapoint.
- Calculate the $p \times p$ dimensional covariance matrix of the zero-mean dataset.
- Calculate the eigenvectors and eigenvalues of this covariance matrix.
- Determine the dimension q of the reduced dataset by looking at the largest eigenvalues. The quality of the reduction depends on how close the sum of the largest q eigenvalues is to the sum of all p eigenvalues.
- Create the $q \times p$ projection matrix E^T from the eigenvectors corresponding to the q largest eigenvalues, and reduce the dataset by multiplying it with this matrix.
- To obtain the corresponding p-dimensional datapoints multiply the new data with the transpose of the projection matrix. Notice that this corresponds to choosing F in (2) such that $F = E$. If q is well chosen these regenerated datapoints should be fairly similar to the original datapoints, thus capturing most of the information in the dataset. Remember to add the mean again when comparing with the original data instead of the zero-mean data.

1.2 Exercises

1.2.1 Redundancy and Random Data

The idea of this exercise is to implement the PCA algorithm in Matlab and apply this to different datasets. The above algorithm can be easily programmed in Matlab with the following functions[1]:

- `cov(X)` calculates the average of the dataset X, subtracts it from each data point, and returns the $p \times p$ covariance matrix of the result. It takes as input an $N \times p$ matrix X with N data points and p the dimensionality of each point. Make sure to check the dimensions of your data matrix and transpose when necessary.
- `diag(X)` returns the diagonal of the square matrix X as a column vector.
- `[V,D]=eigs(X,q)` returns the q largest eigenvalues in the matrix (D) and the $p \times q$ matrix (V) with the corresponding eigenvectors of the square matrix X.
- `transpose(X)` returns the transpose of the matrix X. The command X' has the same effect.
- `help` command will show the available documentation on command.

Generate a 50×500 matrix of Gaussian random numbers (`randn(50,500)`) and try to reduce dimensions with PCA (interpret this as 500 datapoints of dimension 50). Examine different reduced datasets for different dimensions. Try to reconstruct the original matrix. Estimate the error, e.g. by calculating the root mean square difference between the reconstructed and the original data (`sqrt(mean(mean((X-Xhat).^2)))`).

Do the same using the data file `choles_all` (standard in Matlab, can be loaded with `load choles_all`). For the exercise, use only the p component, which is a 21×264 matrix (so the dimensionality of the data equals 21). How does the reduction of random data compare to the reduction of highly correlated data?

1.2.2 Principal Component Analysis on Handwritten Digits

Perform PCA on handwritten images of the digit 3 taken from the US Postal Service database. To access these images, load the Matlab data called `threes.mat` by typing `load threes -ascii`. This loads a 2 megabyte 500×256 matrix called `threes`. Each line of this matrix is a single 16 by 16 image of a handwritten 3 that has been expanded out into a 256 long vector. You can look at the i-th image by typing the command `imagesc(reshape(threes(i,:),16,16),[0,1])`. To have a black-white picture use the command `colormap('gray')` first.

- Compute the mean 3 and display it. Take a look at the command `mean` for this.
- Compute the covariance matrix of the whole dataset of 3s (note that the Matlab function `cov` subtracts the mean automatically, subtracting it beforehand is not incorrect however). Compute the eigenvalues and eigenvectors of this covariance matrix. Plot the eigenvalues (`plot(diag(D)` where D is the diagonal matrix of eigenvalues).
- Compress the dataset by projecting it onto one, two, three, and four principal components. Now reconstruct the images from these compressions and plot some pictures of the four reconstructions.
- Write a function which compresses the entire dataset by projecting it onto q principal components, then reconstructs it and measures the reconstruction error. Note that by choosing how many eigenvectors we use to reconstruct the image we are fixing the number of components, and the quality of the reconstruction. Now call this function for values of q from 1 to 50 (here you probably want to use a loop) and plot the reconstruction error as a function of q.
- What should the reconstruction error be if $q = 256$? What is it if you actually try it? Why?

[1] Matlab also has built-in functions for the PCA algorithm such as `pca(X)` and `processpca(X,maxfrac)` (try for example `maxfrac=0.001`). Be sure to first standardize the data with the function `mapstd(X)`. You can use `processpca('reverse',z,PS)` to get the reconstructed dataset.

- Use the Matlab function `cumsum` to create a vector whose i-th element is the sum of all but the i largest eigenvalues for $i = 1 : 256$. Compare the first 50 elements of this vector to the vector of reconstruction errors calculated previously. What do you notice?

The last question should have shown you a very interesting and important fact: the squared reconstruction error induced by not using a certain principal component is proportional to its eigenvalue. That is why if the eigenvalues fall off quickly then projecting onto the first few components gives small errors because the sum of the eigenvalues that we are not using is not very large.

2 Stacked Autoencoders

2.1 Introduction

An *autoencoder* neural network is an unsupervised learning algorithm that applies backpropagation, setting the target values to be equal to the inputs. I.e., it uses $y(i) = x(i)$.
Figure 1 shows an example of an autoencoder. The autoencoder tries to learn a function $h_{W,b}(x) \approx x$. In other words, it is trying to learn an approximation to the identity function, so as to output $\hat{x}$ that is similar to x. The identity function seems a particularly trivial function to be trying to learn; but by placing constraints on the network, such as by limiting the number of hidden units (called a *sparse* autoencoder), we can discover interesting structure about the data. In fact, this simple autoencoder often ends up learning a low-dimensional representation similar to PCA.

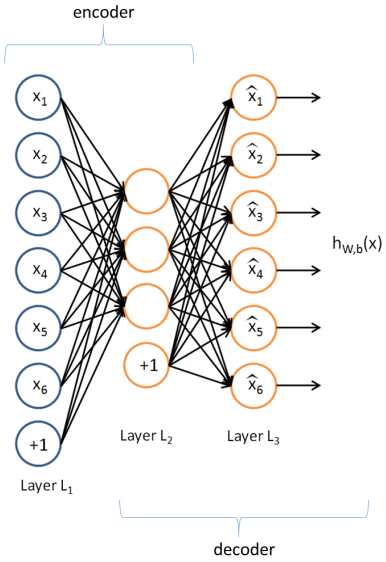

Figure 1: An example of an autoencoder. The part from the input layer to the hidden layer is called the *encoder*. The part from the hidden layer to the output layer is called the *decoder*.

An example of the use of an Autoencoder can be found in [1].

A *stacked autoencoder* is a neural network consisting of multiple layers of sparse autoencoders in which the outputs of each layer are wired to the inputs of the successive layer [2]. The information of interest is contained within the deepest layer of hidden units. This vector gives us a representation of the input in terms of higher-order features. For the use of classification, the common practice is to discard the ”decoding” layers of the stacked autoencoder and link the last hidden layer to a classifier, as depicted in Figure 2.

A good way to obtain good parameters for a stacked autoencoder is to use greedy layer-wise training. So we first train the first autoencoder on the raw input to obtain the weights and biases from the input to the first hidden layer. Then we use this hidden layer as input to train the second autoencoder to obtain the weights and biases from the first to the second hidden layer. Repeat

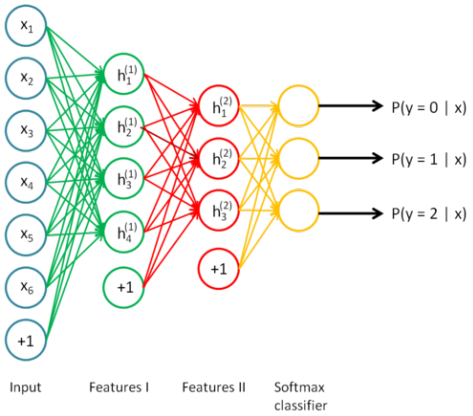

Figure 2: An example of a stacked autoencoder used for classification.

for subsequent layers, using the output of each layer as input for the subsequent layer. This method trains the parameters of each layer individually while freezing parameters for the remainder of the model. To produce better results, after this phase of training is complete, fine-tuning using backpropagation can be used to improve the results by tuning the parameters of all layers at the same time.

A concrete example of a stacked autoencoder can be found in [2].

2.2 Exercise

Run the script `StackedAutoEncodersDigitClassification.m` [3] from Toledo, try to understand what is happening. Use the script `DigitClassification.m` from Toledo to investigate the different parameters (`MaxEpochs`, number of hidden units in each layer, number of layers) and to compare the performance of the Stacked Autoencoder to a normal multilayer neural network. Do not forget to comment the command `rng('default')` when you want to average the results over multiple runs.

Are you able to obtain a better result with different parameters (wrt to the default ones)? How many layers do you need to achieve a better performance than with a normal neural network? What can you tell about the effect of finetuning? Explain *why* and *how* this effects the performance.

3 Convolutional Neural Networks

3.1 Introduction

Convolutional Neural Networks (CNN) is a deep learning technique that uses the concept of *local connectivity*. In a normal multilayer neural network all nodes from subsequent layers are connected, we call these models *fully connected*. The idea is that in a lot of datasets, points that are close to each other, are likely to be a lot more connected than points that are further away. For example, in image datasets where the datapoints represent pixels. Pixels that are close are likely to represent the same part of the image, while pixels that are further away can represent different parts.
CNN's are explained in the Stanford tutorial [4], and also in the following YouTube video through an example [5].

3.2 Exercise

Run the script `CNNex.m` [6] from Toledo, try to understand what is happening[2] .

Take a look at the layers of the downloaded CNN and answer the following questions:

- Take a look at the first convolutional layer (layer 2) and at the dimension of the weights (`size(convnet.Layers(2).Weights)`). If you think back at what you saw in class and/or in [5], what do these weights represent?
- Inspect layers 1 to 5. If you know that a ReLU and a Cross Channel Normalization layer do not affect the dimension of the input, what is the dimension of the input at the start of layer 6 and why?
- What is the dimension of the inputs before the final classification part of the network (i.e. before the fully connected layers)? How does this compare with the initial dimension? Briefly discuss the advantage of CNNs over fully connected networks for image classification.

The script `CNNDigits.m` [7] runs a small CNN on the handwritten digits dataset. Use this script to investigate some CNN architectures. Try out some different amount of layers, combinations of different kinds of layers, dimensions of the weights, etc. Briefly discuss your results. Be aware that some architectures will take a long time to train!

4 Report

Write a report of maximum 4 pages (including text and figures) to discuss the exercises :

- Handwritten Digits PCA and reconstruction
- Digit Classification with Stacked Autoencoders and CNN.
- The answers to the questions in Section 3.2.

References

[1] UFLDL Tutorial: Autoencoders
http://ufldl.stanford.edu/tutorial/unsupervised/Autoencoders

[2] UFLDL Tutorial: Stacked Autoencoders
http://ufldl.stanford.edu/wiki/index.php/Stacked_Autoencoders

[3] MathWorks documentation: Train Stacked Autoencoders for Image Classification
https://nl.mathworks.com/help/nnet/examples/training-a-deep-neural-network-for-digit-classification.html?requestedDomain=www.mathworks.com

[4] UFLDL Tutorial: Convolutional Neural Network
http://ufldl.stanford.edu/tutorial/supervised/ConvolutionalNeuralNetwork

[5] YouTube: How Convolutional Neural Networks work
https://www.youtube.com/watch?v=FmpDIaiMIeA&list=LLoyD7vXkP56-wnw6rR18S6w&index=1

[6] MathWorks documentation: Image Category Classification Using Deep Learning
https://nl.mathworks.com/help/vision/examples/image-category-classification-using-deep-learning.html

[7] MathWorks documentation: Create Simple Deep Learning Network for Classification
https://nl.mathworks.com/help/nnet/examples/create-simple-deep-learning-network-for-classification.html

[2] Since we don not have access to a CUDA-capable GPU on the computers we can not train a large CNN in class. If you are interested in CNNs and you have access to a GPU for computing you may try the full demo [6] at home (note: this is not necessary for the report).