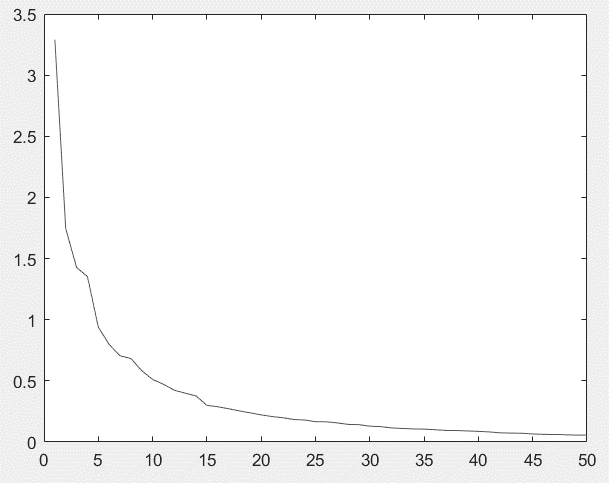
Exercise session 3 : Deep feature learning

**Principal Component Analysis**

Principal component analysis is a method for dimensionality reduction. It projects the data onto only a few principle components while preserving as much of the data’s variance as possible. The principle components are acquired by calculating the eigenvectors of the covariance matrix and selecting the ones with the highest eigenvalues. To analyse the properties of this method, the algorithm as described in the exercise session is programmed in MATLAB.

The algorithm is first performed on a randomly generated dataset and a correlated dataset. The results show that less principle components are needed to reasonably represent the correlated dataset. This is because fewer principle components are needed to capture a large part of the total variation.

Further analysis of the method will be done using the dataset of handwritten images of the digit three. The mean of every pixel over the 500 images is shown in figure 3.1. Figure 3.2 shows a plot of the 50 largest eigenvalues sorted in decreasing order. It can be seen that there is a sharp decrease for the first values and this decrease slows down for the further eigenvalues. After the 50th eigenvalue the values start to go more and more towards zero.

Afbeelding met tekst, binnen, wit, magnetron

Automatisch gegenereerde beschrijving

*Figure 3.1: Visualisation of the mean three. Figure 3.2: Plot of the 50 largest eigenvalues.*

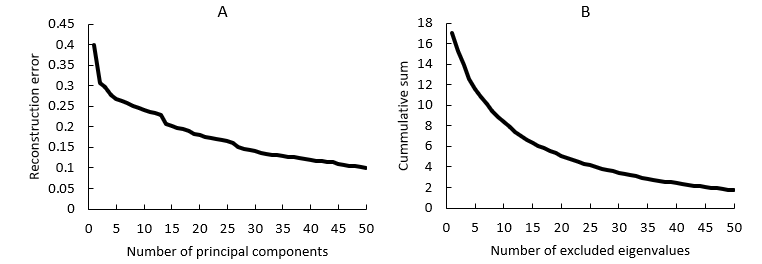
Figure 3.3 shows the reconstructions that can be made using only the most important principal components. These reconstructions are all very similar to the mean three, because that is the best it can do when not enough variation is captured. It can be seen that adding more principle components adds more variety to the images and the visualisations start to divert more and more towards the specific examples.

Afbeelding met tekst

Automatisch gegenereerde beschrijving

*Figure 3.3: Visualisation of the reconstruction of three examples with one to four principal components.*

Figure 3.4.A shows a plot of the reconstruction error in function of the number of principle components used. It can be seen that the reconstruction error decreases by making use of more principal components and this decrease flattens as more principal components get added. Figure 3.4.B shows a plot of the cumulative sum of all but the x largest eigenvalues in function of the number of excluded eigenvalues. The fact that these plots show similar curves indicates that there is a link between the eigenvalues and the reconstruction error. A principle component with a higher eigenvalue captures more of the data’s variation. Not using this principle component leads to a larger reconstruction error. Reasonable data reduction can be performed because the lowest eigenvalues decrease towards zero. Not using these components does therefore not lead to a big increase in the reconstruction error.



*Figure 3.4. A: Plot of the reconstruction error in function of the number of principle components ; B: Cummulative sum of all but the x largest eigenvalues in function of the number of excluded eigenvalues.*

When all 256 principle components are used, there is no data reduction and all the data’s variation can be captured. Therefore it is expected that the reconstruction error is equal to zero. But testing this leads to an actual reconstruction error of 7.43. This can be explained by the floating point rounding errors in the system of MATLAB. These errors are very small and don’t noticeably effect the calculations for this exercise.

**Stacked Autoencoders**

An autoencoder tries to learn the identity function. By placing a constraint on the number of hidden nodes, a lower dimensional representation can be learned. The combination of multiple encoding layers and a classification layer forms the stacked autoencoder. This network can be successfully trained by the combination of greedy layer-wise training and finetuning. To gain insight into the effects of the parameters, multiple tests are performed on the dataset with handwritten digits. All tests are repeated ten times in order to reduce the dependency on the random initialisations.

Varying the ***maximal number of epochs*** shows that enough epochs are needed to be able to capture the relevant features from the data and obtain good accuracies. But if the model is trained for too many epochs, also the noise in the original dataset gets modelled. This leads to lower accuracies due to overfitting. The effect of overfitting seems less severe, but more epochs also take more computational time. So the optimal number of epochs needs to be determined. These findings apply for the encoding and classification layers and also the finetuning step.

For the ***number of hidden units*** in a layer the same trade-off has to be made. Enough units are needed to capture all the relevant information. Using more units leads to a better approximation of the identity function. But using too many units leads to overfitting and increased computational time.

Also the ***number of layers*** can be varied. For every amount of layers, the number of hidden units and the number of epochs need to be optimized accordingly to find the best possible accuracy. By making use of multiple layers, more complex features can be extracted. However, the highest achievable accuracy is about the same for one, two and three layers. The extra layers do not improve the result because the right level of complexity can also be achieved with one layer.

With these insights in mind, it is possible to achieve better results by changing the default parameters in the code. Changing the number of layers is not necessary as explained above. But lowering the number of hidden units and the maximal number of epochs in each layer can prevent the stacked autoencoder from overfitting and therefore perform better on the test set. The increase in the accuracy can be seen in table 3.1.

|  |  |  |
| --- | --- | --- |
|  | No finetuning | With finetuning |
| Default | *84.60* | *99.69* |
| Adjusted | *74.93* | *99.86* |

*Table 3.1: accuracy scores on the test set for the default settings and the adjusted settings*

During the greedy layer-wise training, the parameters are determined layer by layer to get the best possible reconstruction for the inputs of that layer. After the greedy layer-wise training is done, the finetuning step is performed. This is a supervised technique that uses backpropagation to change the parameters of all the layers at the same time. By focussing on the classification instead of reconstruction, the network gets adapted to get better accuracies on the classification task. The results in table 3.1 show that the finetuning step significantly improves the accuracy.

With a high enough number of epochs and hidden units in each layer, the stacked autoencoder can easily obtain an accuracy above 99%. Choosing the right hyperparameters even leads to an accuracy of 99.86%. Normal neural networks can’t achieve this level of accuracy. After trying many architectures with multiple amount of layers and multiple amount of units per layer, the best obtained accuracy is only 97.87%. So, the autoencoders can get better results with even only one encoding layer. Autoencoders perform better because it’s architecture allows for better feature extraction than the normal neural network.

**Convolutional Neural Networks**

A convolutional neural network is a deep learning technique mostly used for image recognition. It consists of different kinds of layers. The most important ones are the convolutional layers and the pooling layers. The convolutional layers serve for finding local patterns. The pooling layers perform feature reduction to prevent overfitting and reduce the computational time. By alternating these layers, relevant features are detected. Finally, the extracted features are used for the classification in the fully connected layers.

The first convolutional layer has 96 filters with dimension 11x11x3. This means that the 96 filters search for features of which both the horizontal and vertical dimension is eleven pixels. The last number in the dimension of the filter represents the colour, which is represented by the RGB values. The weights of these filters represent the features that are extracted from the images. In the first convolutional layer these features are usually quite simple, like edges. In the further convolutional layers, more complex features are possible. These filters are applied on the image and return to what extend the filtered features are present at the local places.

The input images have the dimension 227x227x3. These dimensions are changed in the convolutional layers and the pooling layers. The dimension of the output (O) of each specific layer can be calculated using the following formulas:

Convolutional layer: Pooling layer:

with I the dimension of the input, K the size of the applied filter, P the pooling size and S the stride.

The first convolutional layer has 96 filters with dimension 11x11x3 with stride 4. So this layer also processes the colours in the images. By applying to above formula we get that the output data will have the dimensions 55x55x96. The max pooling layer has dimension 3x3 with stride 2. By applying the above formula we get that the dimensions of the output data will be reduced to 27x27x96. Further calculations for the following layers yield that the dimension of the input before the classification part of the network is 6x6x256. This means that there are finally 256 features extracted and their local presence is shown in a 6x6 grid.

The main motivation behind the architecture of the convolutional neural network is the concept of local connectivity. This means that nodes that are close to each other are more likely to be connected than nodes that are further away. In a convolutional neural network, the feature extraction is done by locally connected layers instead of the standard fully connected layers. This adjustment in the architecture significantly improves the performance of the network for image classification. The convolutional neural network is able to extract better features which are location invariant. It also reduces the computational time because less parameters have to be estimated.

The provided script has been used to investigate the effect of the parameters of the CNN and test some alternative architectures. The insights gained from these tests are discussed below.

Concerning the ***maximal amount of epochs*** it is found that enough epochs are needed to capture all the relevant information. More epochs lead to better feature extraction and better tuning of the classification weights, but training for too long leads to overfitting. Also extracting more ***features in each layer*** can lead to better results, but having to many filters leads to overfitting because less relevant features will be included.

The pooling layers are used for reducing the number of variables. These layers prevent the network from overfitting and speed up the calculations. The higher ***the pooling stride***, the more reduction is performed. But if the stride is too high, some relevant information can get lost. Also the ***pooling dimension*** is important. It has to be wide enough to get some generalization, but the locality needs to be preserved. The digit images have a relatively low number of pixels, so not too much pooling may be done due to the loss of information. The tests also show that both **max and mean pooling** lead to similar results.

The ***dimensionality of the filters*** has to be high enough to be able to extract useful features. But this dimension can’t be too high in order to preserve the advantages from the local connectivity as explained above. When using multiple convolutional layers, it is best to search for more detailed features in the first layer and combine these into more complex features in the subsequent layers.

The ***number of fully connected layers*** and the ***number of nodes per layer*** has to be high enough to make correct classifications using the provided features. But too many layers or too many nodes per layer leads to overfitting. Finally, RELU is the best performing ***activation function*** and the use of batch normalization and cross channel normalisation does not seem the improve the results.

It can be concluded that all these parameters have to be determined empirically for every specific problem. The optimal parameter values ensure that all the relevant information can be captured while also overfitting is prevented.