Questions:

1. In this task you will revisit Lab 2b, (Task1_data.mat). The task is to extend the Hopfield Network into the Stochastic Network. Use the Boltzmann Machine approach to investigate if the results in Lab 2b can be further improved. What values on the parameter T are appropriate? Plot the results before and after the Boltzmann machine approach was introduced. Discuss the results.

2. In this task, you will separate an unknown signal mixture using ICA to recover the original sources (scaling and permutations are in this case irrelevant). The unknown original images $I_1(n_1, n_2)$ and $I_2(n_1, n_2)$ have been mixed by an unknown mixture matrix $\mathbf{A}$ (size 2×2) into two signal mixtures $J_1(n_1, n_2)$ and $J_2(n_1, n_2)$ according to:

$$J_1(n_1, n_2) = A_{11} I_1(n_1, n_2) + A_{12} I_2(n_1, n_2)$$

$$J_2(n_1, n_2) = A_{21} I_1(n_1, n_2) + A_{22} I_2(n_1, n_2)$$

   You will demix these signals using the demixing matrix $\mathbf{W}$ (size 2×2) according to:

$$S_1(n_1, n_2) = W_{11} J_1(n_1, n_2) + W_{12} J_2(n_1, n_2)$$

$$S_2(n_1, n_2) = W_{21} J_1(n_1, n_2) + W_{22} J_2(n_1, n_2)$$

   Data

   In the Matlab data file Task2_data.mat there are two grayscale images, each of size 200×250 pixels (picture elements). These images are the observed signal mixtures J_1 and J_2, as illustrated below.

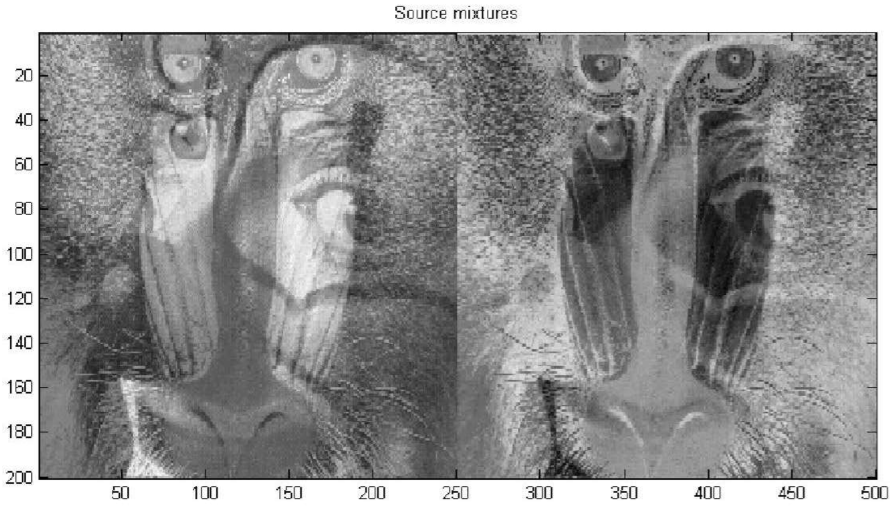


Figure 1: *Observed source mixtures; J_1 on the left side and J_2 on the right side.*

   The task

   The objective of this task is to separate the source mixtures so as to recover the original images (image scaling and permutation is in this case irrelevant).

In comparison to ICA algorithm presented in the lecture (1D time signals), here we deal with 2D spatial signals. Thus when reviewing the ICA algorithm in the lecture notes, the following changes need to be considered:

Stack the mixed sources into a matrix **U** of size 2 x 200*250.

$$\mathbf{S} = \mathbf{W}^{(k)}\mathbf{U}$$

$$\mathbf{W}^{(k+1)} = \mathbf{W}^{(k)} + \alpha(\mathbf{I} - f(\mathbf{S})\mathbf{S}^T/N)\mathbf{W}^{(k)}$$

where **S** and **U** are matrices NOT vectors, and $N = 200 * 250 = 50000$. Assume the tanh-nonlinearity, i.e., $f(\mathbf{s}) = tanh(\mathbf{s})$. Implement a suitable stopping condition!

How many iterations are required in order to clearly see the original sources without any dominant artifacts from the other source? How sensitive is the algorithm with regards to convergence and choice of the step size? Plot the separated sources.

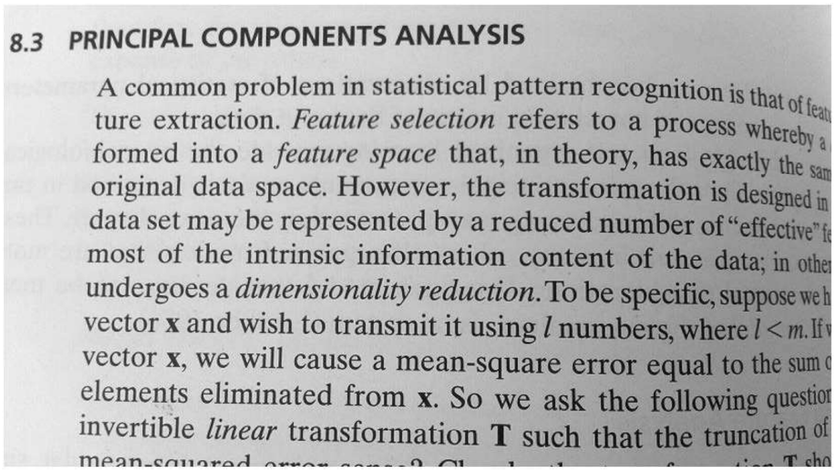
Hints:
You can draw an image in matlab using the commands:
figure;
imagesc(S1)
colormap gray

It may be that one of the separated outputs are inverted. This is ok, since the ICA solution is scale invariant. You may simply multiply the separated output with -1 to correct for this!

3. In this task, you will compress image data using principal component analysis, Generalized Hebbian Algorithm (GHA) and non-linear signal compression. Compression is analogous to preserving key features in assessed by directions (eigenvectors) that maximize variance (eigenvalues).

   Data

   In the Matlab data file Task3_data.mat is an grayscale image used for this task. The image is a photo from chapter 8.3 of a well known text book in neural network theory.


8.3 PRINCIPAL COMPONENTS ANALYSIS

A common problem in statistical pattern recognition is that of feat
ture extraction. *Feature selection* refers to a process whereby a
formed into a *feature space* that, in theory, has exactly the sam
original data space. However, the transformation is designed in
data set may be represented by a reduced number of "effective" fe
most of the intrinsic information content of the data; in other
undergoes a *dimensionality reduction.* To be specific, suppose we h
vector **x** and wish to transmit it using *l* numbers, where $l < m$. If v
vector **x**, we will cause a mean-square error equal to the sum c
elements eliminated from **x**. So we ask the following questior
invertible *linear* transformation **T** such that the truncation of
mean-squared error

Figure 2: *The image used in this task.*

Task 3a

- Divide the image into non-overlapping blocks of 8×8 pixels each. There should be 57×102 non-overlapping blocks for this image.
- Vectorize each block by stacking the columns of each block ontop of each other to a 64×1 column vector, $\mathbf{x}_b$. where $b = 1...5814$ is the block index.
- Compute the auto-correlation matrix of the image

$$\mathbf{R} = \frac{1}{5814} \sum_{b=1}^{5814} \mathbf{x}_b \mathbf{x}_b^T$$

  (R will have the size 64×64)
- Compute the principal components of the correlation matrix. There will be 64 principal components $\mathbf{d} = (\mathbf{d}_1, ..., \mathbf{d}_{64})^T$ (sorted eigenvalues) and 64 corresponding eigenvectors $\mathbf{V} = (\mathbf{v}_1, ..., \mathbf{v}_{64})$. Note that, the principal components are really the eigenvalues sorted in descending order, i.e., $d_{64} \leq ... \leq d_2 \leq d_1$. Hint: use the **eig** command in Matlab.
- Now select the M (where $1 \leq M \leq 64$) first principal components. This corresponds to selecting the first M columns of $\mathbf{V}$, i.e., $\mathbf{Q} = \mathbf{V}(:, m)$. $\mathbf{Q}$ is referred to as our compression matrix, and it will be used for all input blocks in the input image.
- The compression is such that the input signal blocks are projected onto the eigenvectors corresponding to the M first principal components. The compressed image will have the size $57 \times 102 \times M$. Hint: each block will be individually compressed to a corresponding block in the output image according to (Matlab notation!): $\mathbf{y}_b = \mathbf{Q}.' * \mathbf{x}_b$, where $b = \{1, ..., 5814\}$.
- Reconstruct all compressed image blocks by re-projecting the compressed data blocks onto the principal component (eigen) vectors: $\mathbf{u}_b = \mathbf{Q} * \mathbf{y}_b$ (each block vector should have the size 64×1). Hint: you can do the compression and reconstruction at once: $\mathbf{u}_b = \mathbf{Q} * \mathbf{Q}.' * \mathbf{x}_b$.

- Now, reshape the block vector $\mathbf{u}_b$ by ordering the columns beside each other. The reconstructed block will have the size 8×8 again. Construct the output image $\mathbf{U}$ by inserting all reconstructed 8×8-blocks into the output image. The output image will again have the size 456×816 pixels.
- The mean square error of the compression is computed as:

$$E_{MSE} = 10 * log_{10} \frac{\sum_{i=1}^{456} \sum_{j=1}^{816} (X_{i,j} - U_{i,j})^2}{\sum_{i=1}^{456} \sum_{j=1}^{816} X_{i,j}^2}$$

  where $X_{i,j}$ and $U_{i,j}$ are pixel values of the input image and reconstructed output image, respectively.
- The compression ratio (in percent) is computed as:

$$C(M) = 100 * (1 - \frac{M}{64})$$

Plot the compression ratio towards the mean square error (E_{MSE} on y-axis and compression ratio on x-axis). Using your subjective experience, what is the highest possible compression ratio before the image becomes unreadable? How many components M are required to have less than -17 dB mean square error?

Task 3b

Use the GHA to find M PCA components that have less than -17 dB mean square error (same M as in Task 3a). Note that in previous task you calculated PCA components directly, thus E_{MSE} from GHA solution will be higher. How many iterations are needed to get similar E_{MSE} as in previous task? How close can you get? Print the reconstructed image with PCA components approximated with GHA and compare with reconstructed image with PCA components from previous task.

Task 3c

In this task a 2-layer feed-forward neural network with back-propagation algorithm is used for nonlinear signal compression. The following structure is used:

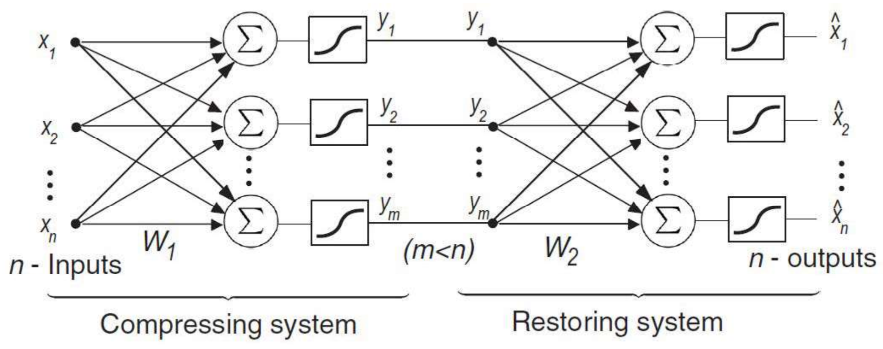


Figure 3: *The non-linear signal compression model.*

Use the bi- or uni-polar sigmoid function. Chose $(\mathbf{x}_1, \mathbf{x}_1), (\mathbf{x}_2, \mathbf{x}_2), ..., (\mathbf{x}_N, \mathbf{x}_N)$ as a training pairs. This implies that weight matrix $\mathbf{W}_1$ is trained for compression of a signal and $\mathbf{W}_2$ to restore it. The dimension $n = 64$ and $m = M$, same as in previous tasks. What E_{MSE} can you obtain with the non-linear compression method? How many iteration does it take?

4. In this task, you will experiment with the back-propagation algorithm on data from a real product and using measured data! This experiment is taken from an application in industry!

   The system

   The experiment deals with an alarm device. The intention of the alarm device is to detect when a system is being manipulated, e.g., during thievery. When the system is being manipulated, an alarm should be set off. Electro-magnetic field strength measurements on the system are used to detect manipulations. If the system is being manipulated, there are changes in the EM-field strength.

   The training database

   Each measurement comprises of 16 EM-field strength data points together with one target sample value, see Task4_data.mat. If the $target = +1$ it implies everything is ok, whereas if $target = -1$, it means that the system is being manipulated. A database of measurements and targets is provided:

   - Input : size 16×371, contains 371 EM-measurement vectors
   - Target : size 1×371, with the value $+1$ if the system is OK, -1 if the system is being manipulated.

   Pre-processing

   We assume that it is not the actual EM field strength value that is of importance, instead the shape of the 16 measurement points. Hence, before starting the training, each data vector needs to be pre-processed. There is a plethora of pre-processors available. We will use a simple pre-processor on each input vector $\mathbf{Input}_k$ of size 16×1 :

   $$\mathbf{Input}_k = (2*(\mathbf{Input}_k - min(\mathbf{Input}_k))/(1 + max(\mathbf{Input}_k) - min(\mathbf{Input}_k))) - 1$$

   This pre-processor yields that the vector's values lie approximately between -1 and $+1$. The functions $min()$ and $max()$ find the minimal resp. maximal element of a vector.

   Evaluation measures

   We will estimate the following evaluation measures:

   - P(true positive): Probability that output ≥ 0 given that Target = +1;
   - P(false positive): Probability that output < 0 given that Target = +1;
   - P(true negative): Probability that output < 0 given that Target = -1;
   - P(false negative): Probability that output ≥ 0 given that Target = -1;
   - P(true alarm) = P(true positive) + P(true negative)

- P(false alarm) = P(false positive) + P(false negative)

Probabilities are approximated by counting the rate of probability, e.g.,

- P(true positive) = Count(Output ≥ 0 given Target = +1) / Count(Target = + 1)
- P(false positive) = 1 - P(true positive)
- P(true negative) = Count(Output < 0 given Target = -1) / Count(Target = - 1)
- P(false negative) = 1 - P(true negative)

Task

- Implement the data pre-processor.
- Implement a multilayer feed forward neural network and train it using the back-propagation algorithm. Use the bipolar sigmoid non-linearity.
- Use K-Fold learning and implement strategies to improve generalization and minimize over-fitting, e.g., by using early stopping.
- Plot the false alarm rates P(false positive) and P(false negative) during training epochs.
- What is an optimal network structure? I.e., number of hidden layers, number of neurons per layer? What number of iterations are required before stopping? What is the final false alarm rates?

5. The Redial Basis Function Networks (RBFN) can be used both for function approximation and pattern classification instead of Multi-layers Feed Forward Networks (MLFFN). In this task you shall redo the previous task this time by using RBF networks. Use the Kohonen Network in the first part and linear layer of neurons in the second part. Due to clustered input space you will have to use the Self-Organizing Feature Map (SOFM) Algorithm for training the Kohonen Network.

   Compared to previous task what is an optimal network structure? I.e., number of neurons in kohonen network? What number of iterations are required before stopping? Is the convergence faster then in previous task? What is the final false alarm rates? Can you get it better then in previous task?

Good Luck!