Statistical Machine Learning: Assignment #4

Due on Monday, 18 January 2016 at 15:45

W.A.J.J. Wiegerinck, T.M. Heskes, T. Claassen, H.J. Kappen

S. Lederer (1234567), P.W.J. Ebel (4399803)

(1)

When considering the learning of weights of neural networks, it is important to keep in mind that the learning procedure might get stuck in local extrema or saddle points of the cost function. Depending on the initial point from which the weight space is to be explored (that is: the initial weight vector), there might be a variable number of such critical points in the direct neighborhood that the local learning procedure might get stuck in. For this reason, the initial value of weights might in general have an important effect on the outcome of learning the weights. When we focus on a multilayer perceptron (MLP) and the backpropagation algorithm treated in the lecture as an example of such a learning procedure, it is not a good idea to start with a network in which all the weights have been initialized to zero. When all weights are initialized to zero, every neural unit in the network gets a total zero input drive and, assuming hyperbolic tangent (tanh) activity functions in the input- and hidden layer(-s), each of these unit will be active at a firing rate of 0.5. Assuming that the output unit has a linear activation function (chosen for a regression task), its zero input drive results in a firing rate of 0 - which will be the overall output of the MLP network. Moreover, for all weights $w_{kj} = 0$, the Backpropagation formula given in Bishop eq. (5.56)

$$\delta_j = h'(a_j) \sum_k w_{kj} \delta_k$$

always evaluates to zero. Consequently, the error-derivatives in the weights, which are given by the back-propagated errors δ_j multiplied by the previous layer's activity (Bishop eq. (5.53))

$$\frac{\partial E_n}{\partial w_{ji}} = \delta_j x_i$$

will also be zero. Thus no change of weights occurs and the weights will keep their initial values, all being zero.

Problem 1

(2)

Given the network architecture, network parameters and input data specified in the exercise, one full training cycle of the network is given by:

1) forward propagation of activations, use Bishop eq. (5.48) and (5.49):

1.1) calculate hidden layer activities:

$$\begin{aligned} z_1 &= tanh(\sum_i w_{1,i}^{(1)} \cdot x_i + b_1^{(1)}) \\ &= tanh(1 \cdot 0.5 + 1) \\ &= tanh(1.5) \\ &\approx 0.905 \end{aligned}$$

Problem 1 continued on next page...

$$\begin{aligned} z_2 &= tanh(\sum_i w_{2,i}^{(1)} \cdot x_i + b_2^{(1)}) \\ &= tanh(0.1 \cdot 0.5 + 0) \\ &= tanh(0.05) \\ &\approx 0.05 \end{aligned}$$

$$\begin{aligned} z_3 &= tanh(\sum_i w_{3,i}^{(1)} \cdot x_i + b_3^{(1)}) \\ &= tanh(-1 \cdot 0.5 + 1) \\ &= tanh(0.5) \\ &\approx 0.462 \end{aligned}$$

1.2) calculate output layer activity:

$$\begin{aligned} y &= \sum_j w_{1,j}^{(2)} \cdot z_j + b_1^{(2)} \\ &= -1 \cdot tanh(1.5) + 0.1 \cdot tanh(0.05) - 1 \cdot tanh(0.5) + 2 \\ &\approx 0.638 \end{aligned}$$

2) calculate output layer errors:

$$\begin{aligned} \delta_k &= y - T \\ &= y - 0.25 \\ &\approx 0.388 \end{aligned}$$

the Sum-of-Squares Error (SSE) is given by :

$$\begin{aligned} E(t) &= \frac{1}{2} \cdot \delta_k^2 \\ &\approx 0.075 \end{aligned}$$

3) back propagation of errors, use Bishop eq. (5.56):

$$\begin{aligned} \delta_1 &= (1 - z_1^2) \cdot \sum_k w_{k,1}^{(2)} \delta_k \\ &= (1 - z_1^2) \cdot (-1 \cdot \delta_k) \\ &\approx -0.07 \end{aligned}$$

$$\begin{aligned} \delta_2 &= (1 - z_2^2) \cdot \sum_k w_{k,2}^{(2)} \delta_k \\ &= (1 - z_2^2) \cdot (0.1 \cdot \delta_k) \\ &\approx 0.039 \end{aligned}$$

Problem 1 continued on next page. . .

$$\begin{aligned}\delta_3 &= (1 - z_3^2) \cdot \sum_k w_{k,3}^{(2)} \delta_k \\ &= (1 - z_3^2) \cdot (-1 \cdot \delta_k) \\ &\approx -0.305\end{aligned}$$

4) evaluate the error derivatives, use Bishop eq. (5.53):

4.1) calculate second layer error derivatives:

$$\begin{aligned}\frac{\partial E}{\partial w_{1,1}^{(2)}} &= \delta_k \cdot z_1 \\ &\approx 0.351\end{aligned}$$

$$\begin{aligned}\frac{\partial E}{\partial w_{1,2}^{(2)}} &= \delta_k \cdot z_2 \\ &\approx 0.019\end{aligned}$$

$$\begin{aligned}\frac{\partial E}{\partial w_{1,3}^{(2)}} &= \delta_k \cdot z_3 \\ &\approx 0.179\end{aligned}$$

$$\begin{aligned}\frac{\partial E}{\partial w_{1,0}^{(2)}} &= \delta_k \cdot 1 \\ &\approx 0.388\end{aligned}$$

4.2) calculate first layer error derivatives:

$$\begin{aligned}\frac{\partial E}{\partial w_{1,1}^{(1)}} &= \delta_1 \cdot x \\ &\approx -0.035\end{aligned}$$

$$\begin{aligned}\frac{\partial E}{\partial w_{2,1}^{(1)}} &= \delta_2 \cdot x \\ &\approx 0.02\end{aligned}$$

$$\begin{aligned}\frac{\partial E}{\partial w_{3,1}^{(1)}} &= \delta_3 \cdot x \\ &\approx -0.153\end{aligned}$$

Problem 1 continued on next page. . .

$$\begin{aligned}\frac{\partial E}{\partial w_{1,0}^{(1)}} &= \delta_1 \cdot 1 \\ &\approx -0.07\end{aligned}$$

$$\begin{aligned}\frac{\partial E}{\partial w_{2,0}^{(1)}} &= \delta_2 \cdot 1 \\ &\approx 0.039\end{aligned}$$

$$\begin{aligned}\frac{\partial E}{\partial w_{3,0}^{(1)}} &= \delta_3 \cdot 1 \\ &\approx -0.305\end{aligned}$$

5) update the weights, use Bishop eq. (5.43):

5.1) update weights of connections to the output layer:

$$\begin{aligned}w_{1,1}^{(2)}(t+1) &= w_{1,1}^{(2)}(t) - \eta \cdot \nabla E(w_{1,1}^{(2)}(t)) \\ &= -1 - 0.5 \cdot \delta_k \cdot z_1 \\ &\approx -1.176\end{aligned}$$

$$\begin{aligned}w_{1,2}^{(2)}(t+1) &= w_{1,2}^{(2)}(t) - \eta \cdot \nabla E(w_{1,2}^{(2)}(t)) \\ &= 0.1 - 0.5 \cdot \delta_k \cdot z_2 \\ &\approx 0.09\end{aligned}$$

$$\begin{aligned}w_{1,3}^{(2)}(t+1) &= w_{1,3}^{(2)}(t) - \eta \cdot \nabla E(w_{1,3}^{(2)}(t)) \\ &= -1 - 0.5 \cdot \delta_k \cdot z_3 \\ &\approx -1.09\end{aligned}$$

$$\begin{aligned}w_{1,0}^{(2)}(t+1) &= w_{1,0}^{(2)}(t) - \eta \cdot \nabla E(w_{1,0}^{(2)}(t)) \\ &= 2 - 0.5 \cdot \delta_k \cdot 1 \\ &\approx 1.806\end{aligned}$$

5.2) update weights of connections to the hidden layer:

$$\begin{aligned}w_{1,1}^{(1)}(t+1) &= w_{1,1}^{(1)}(t) - \eta \cdot \nabla E(w_{1,1}^{(1)}(t)) \\ &= 1 - 0.5 \cdot \delta_1 \cdot x \\ &\approx 1.018\end{aligned}$$

Problem 1 continued on next page. . .

$$\begin{aligned} w_{2,1}^{(1)}(t+1) &= w_{2,1}^{(1)}(t) - \eta \cdot \nabla E(w_{2,1}^{(1)}(t)) \\ &= 0.1 - 0.5 \cdot \delta_2 \cdot x \\ &\approx 0.09 \end{aligned}$$

$$\begin{aligned} w_{3,1}^{(1)}(t+1) &= w_{3,1}^{(1)}(t) - \eta \cdot \nabla E(w_{3,1}^{(1)}(t)) \\ &= -1 - 0.5 \cdot \delta_3 \cdot x \\ &\approx -0.924 \end{aligned}$$

$$\begin{aligned} w_{1,0}^{(1)}(t+1) &= w_{1,0}^{(1)}(t) - \eta \cdot \nabla E(w_{1,0}^{(1)}(t)) \\ &= 1 - 0.5 \cdot \delta_1 \cdot 1 \\ &\approx 1.035 \end{aligned}$$

$$\begin{aligned} w_{2,0}^{(1)}(t+1) &= w_{2,0}^{(1)}(t) - \eta \cdot \nabla E(w_{2,0}^{(1)}(t)) \\ &= 0 - 0.5 \cdot \delta_2 \cdot 1 \\ &\approx -0.019 \end{aligned}$$

$$\begin{aligned} w_{3,0}^{(1)}(t+1) &= w_{3,0}^{(1)}(t) - \eta \cdot \nabla E(w_{3,0}^{(1)}(t)) \\ &= 1 - 0.5 \cdot \delta_3 \cdot 1 \\ &\approx 1.153 \end{aligned}$$

6) re-evaluate forward propagation of activations, calculate error at output layer and compare to previous value in 2):

6.1) calculate hidden layer activities:

$$\begin{aligned} z_1 &= tanh(\sum_i w_{1,i}^{(1)} \cdot x_i + b_1^{(1)}) \\ &= tanh(1.018 \cdot 0.5 + 1.035) \\ &\approx 0.913 \end{aligned}$$

$$\begin{aligned} z_2 &= tanh(\sum_i w_{2,i}^{(1)} \cdot x_i + b_2^{(1)}) \\ &= tanh(0.09 \cdot 0.5 - 0.02) \\ &\approx 0.026 \end{aligned}$$

Problem 1 continued on next page. . .

$$\begin{aligned} z_3 &= tanh(\sum_i w_{3,i}^{(1)} \cdot x_i + b_3^{(1)}) \\ &= tanh(-0.924 \cdot 0.5 + 1.153) \\ &\approx 0.599 \end{aligned}$$

6.2) calculate output layer activity:

$$\begin{aligned} y &= \sum_j w_{1,j}^{(2)} \cdot z_j + b_1^{(2)} \\ &- 1.176 \cdot 0.913 + 0.09 \cdot 0.026 - 1.09 \cdot 0.599 + 1.806 && \approx 0.0817 \end{aligned}$$

6.3) calculate output layer errors:

$$\begin{aligned} \delta_k &= y - T \\ &= y - 0.25 \\ &\approx -0.168 \end{aligned}$$

the Sum-of-Squares Error (SSE) is given by :

$$\begin{aligned} E(t+1) &= \frac{1}{2} \cdot \delta_k^2 \\ &\approx 0.014 \end{aligned}$$

The update has improved the network output quality, given by a reduction in the SSE E, as we can see since $E(t+1) < E(t)$.

(3)

The nonlinearity of the activation functions for the hidden nodes is crucial for a network to learn more than mainly trivial regression or classification problems. Assume that (a) the activation functions for the hidden units are linear. It is a fact that (b) the summation over linear mappings (as e.g. one layer feeding an input drive forward to a neuron in the next layer) and compositions of linear mappings (as e.g. activations projecting from one layer to the next and being mapped again, linearly) give again a linear mapping. Putting (a) and (b) together, the network could only solve a linear regression problem (or: linearly separable classification problems). This network would be of very limited use and is in fact equivalent to a constrained architecture not having any hidden units: assuming linear hidden units, the mapping performed by the hidden layer is again linear and thus could as well be computed by the linear output unit (without any inputs from hidden units), as argued above.
The network considered further in this exercise has hidden nodes with tanh activation functions. The hyperbolic tangent is of a sigmoidal shape whose essential feature is to squash the output in the interval $]-1, +1[$ which is the image of the selected activity function. That is, outputs are bound from below by -1 and saturate to $+1$. The universal approximation theorem (see e.g. Cybenko (1989) and Hornik (1989)) guarantees that networks having at least one hidden layer of units with tanh or similarly behaving activation functions can learn to approximate any computable function with arbitrary precision, given a sufficient number of

Problem 1 continued on next page. . .

hidden units.

(4)

Given a data set of 50 data points x_i, t_i sampled over the interval [1, 1] from a noisy quadratic function: $t = x^2 + \epsilon$, with $\epsilon \sim N(\epsilon|0, 0.2)$. If sequential or batch updating of weights converges, depends on whether the applied learning rate is chosen sufficiently small and if the employed cost function is bounded from below (this theorem is proven on SML lecture slide 272). Given a SSE cost function and small enough learning rates, we thus assume that both updating schemes converge - but the points they converge to are not necessarily identical: for a gradient descent updating of weights, batch mode might run into a point in weight space that is a local minimum with respect to the whole data set. We think that sequential updating would be unlikely to get stuck in the very same point, as a vanishing gradient over the whole batch of data does not necessarily imply that also all gradients over each single data point are zero. Besides being more robust to getting stuck in local minima, we also believe (based on: DR Wilson, TR Martinez (2003)) that sequential updating might allow for a more efficient exploration of weight space than batch mode: whereas the latter only takes a single gradient per cycle, and takes a step in a straight line, sequential updating takes many steps per cycle and thus can easier follow curves on the error surface. The single big step might require to reduce the learning rate at later stages of the learning procedure in order for it to remain stable, whereas the many small steps of sequential updating allow for larger learning rates. All in all, we believe that both sequential and batch updating can be guaranteed to converge, but batch mode suffers some drawbacks that a sequential updating scheme might avoid.

Problem 2

(1)

The data, sampled from an isotropic 2D Gaussian $Y = 3 \cdot N(X|\frac{2}{5}I_2)$ depict in Fig. 1, is obtained via

```
[x,y]  = meshgrid(-2:0.1:2,-2:0.1:2);
X      = [x(:), y(:)];
Y      = 3*mvnpdf(X, [0 0], 2/5 * eye(2));
```

(2)

A 2D plot of the initial output of the network over the same $[2,2]\times[2,2]$ domain X as the Gaussian in exercise (1) is depict in Fig. 2 obtained by calling the implemented MLP (code given below, it is well documented - so please see the comments for explanation)

```
function y_test = myMLP(M,X,Y,cycles,eta,permIdx)
% myMLP simulates a multilayer perceptron
%   M        - number of units in the hidden layer
%   X        - input values of data
%   Y        - target values of data
%   cycles   - number of times to loop through the training data
%   eta      - learning rate
%   mom      - momentum term (for gradient descent)

%% initialize the MLP

mom = 0;                          % set momentum to zero
D = size(X,2);                    % dimension of input
X = [X,ones(length(X),1)];        % concatenate 1's for bias input
D = D + 1;                        % incr. dimension of input (hidden layer bias)
M = M + 1;                        % incr. dimension of hidden layer (output bias)

```

Problem 2 continued on next page...

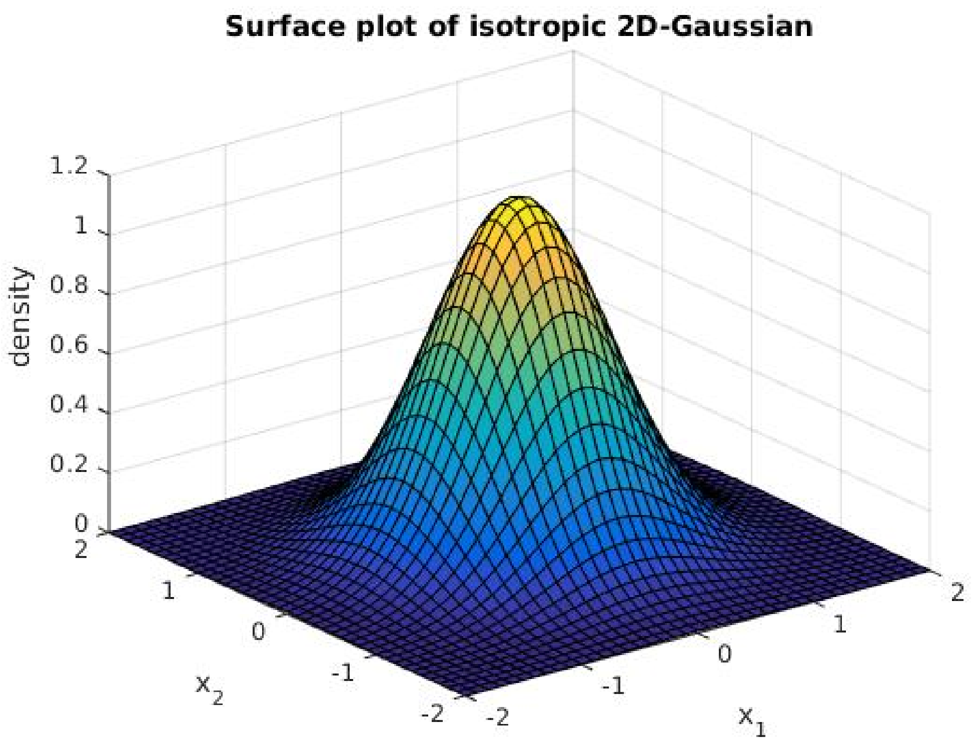


Figure 1: Plot of the isotropic 2D Gaussian $Y = 3 \cdot N(X|0, \frac{2}{5}I_2)$.

```
% initialize weights in [-0.5,0.5] uniformly
w1=rand(D,M)-0.5;                   % weights of input -> layer1
w1(:,M) = 0;                        % (the weight to the bias is irrelevant)
w2=rand(M,1)-0.5;                   % weights of layer1 -> output

y_train = zeros(1,cycles*size(X,1));  % declare space to store development of outputs
E_train = zeros(1,cycles);            % declare space to store development of cost
dw_train = zeros(1,cycles);           % declare space to store development of change in
    weights

dw1 = zeros(M,D);   % initialize delta_weights layer 1
dw2 = zeros(1,M);   % initialize delta_weights output layer

% store outputs per cycles
y_cycle = zeros(1,size(X,1));
E_cycle = zeros(1,cycles);

for loops = 1:cycles       % loop through all cycles
    idx = 1;               % initialize/ reset number of iterations
    while (idx<=size(X,1))
        %% [forward propagate] activities, calculate error and cost

        %seq = randi(length(X));            % randomly choose input for stoch.grad.desc.
        seq = idx;

        a1 = X(seq,:)*w1; z  = tanh(a1);  % use tanh activation fct. for hidden units
        z(1,M) = 1;                         % set manually added hidden unit to 1 (bias)
        a2 = z*w2;          y  = a2;        % use linear activation fct. for output unit

        error = y - Y(seq,:);                % obtain errors and cost E (SSE)
        E     = 0.5 * sum(error.*error);     % at the output layer

        %% [back propagate] errors and calculate derivatives
        d2 = error;                        % back propagate to hidden layer, eq. (5.65)
        d1 = (1-z.^2)' .* (d2*w2);         % back propagate to input layer, eq. (5.66)

        dE2 = d2*z;                        % get error derivative, Bishop eq (5.53),
        dE1 = d1*X(seq,:);                 % analogous to Bishop eq (5.67)

        dw2 = eta * dE2 + mom * dw2;       % update weights, for mom = 0 this is
        dw1 = eta * dE1 + mom * dw1;       % equivalent to Bishop eq (5.41)

        %% calculate update in weights (set mom=0 for Bishop eq. (5.43))
        w2 = w2 - dw2';
        w1 = w1 - dw1';

        % update and store variables per learning step over all cycles
      y_train((loops-1)*size(X,1)+idx) = y;          % store regression outputs

        % update and store variables per learning step for each cycle
        dw  = max(max(max(abs(dw1))),max(abs(dw2))); % calculate maximum change of weights
        dw_train(loops)= dw_train(loops)+dw;          % store maximum adjustment of weights
        E_train(loops) = E_train(loops) + E;          % store cost (add up over all inputs)

        idx = idx+1;                                  % increment iteration counter
    end

        % feed forward the whole data set to get data for the plots
        for idx = 1:size(X,1)
            a1 = X(idx,:)*w1;
            z = tanh(a1); z(1,end) = 1;     % get hidden layer act.
            a2 = z*w2;                      % get output layer act.
            y_cycle(idx) = a2;
        end
        y_plotti = y_cycle;

        error = y_cycle' - Y;                         % obtain errors and cost E (SSE)
        E_cycle(loops)= 0.5 * sum(error.*error);      % at the output layer

        % every 20th cycle, do a surface plot
        if mod(loops,10) == 0
            % print the current error after each completed cycle
            fprintf('cycle %d; error %f; dw %f\n',loops,E_cycle(loops),dw_train(loops));

            if exist('permIdx','var')
                % un-permutate the outputs of the network to plot them in order
                [~,unperm] = sort(permIdx);
                y_plotti = y_cycle(unperm)';
            end

            %figure();
            [x,y] = meshgrid(-2:0.1:2,-2:0.1:2);
            surf(x,y,reshape(y_plotti, size(x)))
            xlabel('x_1'); ylabel('x_2'); zlabel('density');
            title(strcat('Outputs of MLP trained for', num2str(loops),'cycles'))
            drawnow
        end

end

%% do testing
```

Problem 2 continued on next page. . .

```

y_test = nan(1,size(X,1));  % store regression outputs
E_test = nan(1,size(X,1));
jdx = 1;                    % initialize number of iterations

while (jdx<=length(X))
    %% [forward propagate] activities, calculate error and cost

    %seq = randi(length(X));            % randomly choose input for stoch.grad.desc.
    seq = jdx;

    a1 = X(seq,:)*w1; z  = tanh(a1);  % use tanh activation fct. for hidden units
    z(1,M) = 1;                         % set manually added unit to 1 (bias)
    a2 = z*w2;          y  = a2;        % use linear activation fct. for output unit

    error = y - Y(seq,:);               % obtain errors and cost E (SSE)
    E     = 0.5 * sum(error.*error);    % at the output layer

    % update and store variables
    y_test(jdx) = y;                              % store regression outputs
    E_test(jdx) = E;                              % store cost

    jdx = jdx+1;
end

%% return statistics and plot figures

if cycles > 0 % if training occured, plot training statistics
    figure()
    %loglog(E_cycle)
    plot(E_cycle)
    title(strcat('Training error as a function of cycles, final:',num2str(E_cycle(end))))
    xlabel('cycle')
    ylabel('training error (SSE)')

    figure()
    %loglog(dw_train)
    plot(dw_train)
    title(strcat('Change in weights as a function of cycles, final:',num2str(dw_train(end)))
    )
    xlabel('cycle')
    ylabel('max. weight change')
    fprintf('train error: %f \n',E_train(end));
    fprintf('dw: %f \n',dw_train(end));
end
end
```

with inputs X and no adjustment of weights and biases after initialization (this is equivalent to setting the learning rate η or the number of learning cycles to zero).

```
% please see mlp.m

% initialize parameters with which to call the MLP
hUnits = 8;         % specify number of hidden units
eta = 0.1;          % learning rate
cycles = 0;         % maximal number of cycles over training set

% call the MLP
y_test = myMLP(hUnits,X,Y,cycles,eta);
```

(3)

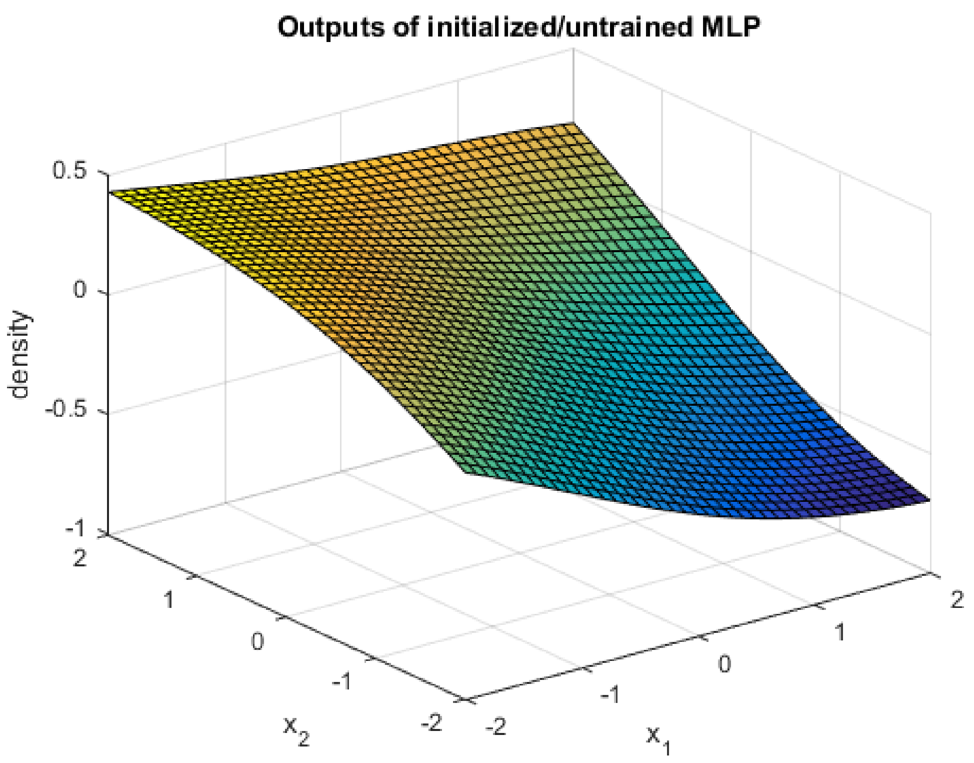


Figure 2: 2D plot of the initial output of the untrained network over X.

The network is initialized with 8 hidden units and called to train for 500 full cycles on the data (X,Y) defined in exercise (1).

```
% initialize parameters with which to call the MLP
hUnits = 8;          % specify number of hidden units
eta = 0.01;          % learning rate
cycles = 500;        % maximal number of cycles over training set

% call the MLP
y_test = myMLP(hUnits,X,Y,cycles,eta);
```

The results in terms of the development of SSE and the resulting output plot are depict in Figures 3 and 4, respectively. Fig. 5 shows a plot of the network output after 200 training cycles. One can observe that a coarsely Gaussian-shaped function emerges, that gets more refined features in Fig. 4 after 500 cycles. All in all, the shape of the target function can be recognized in the network output, but the approximation is far from perfect and convergence to the true distribution is fairly slow.

(4)

The data (X,Y) is shuffled and then the MLP is called with the parameters declared in exercise (3) on the randomly permuted data set.

```
% permute X and Y to a random order
permIdx = randperm(length(X));
X = X(permIdx,:);
Y = Y(permIdx);

% call the MLP
```

Problem 2 continued on next page. . .

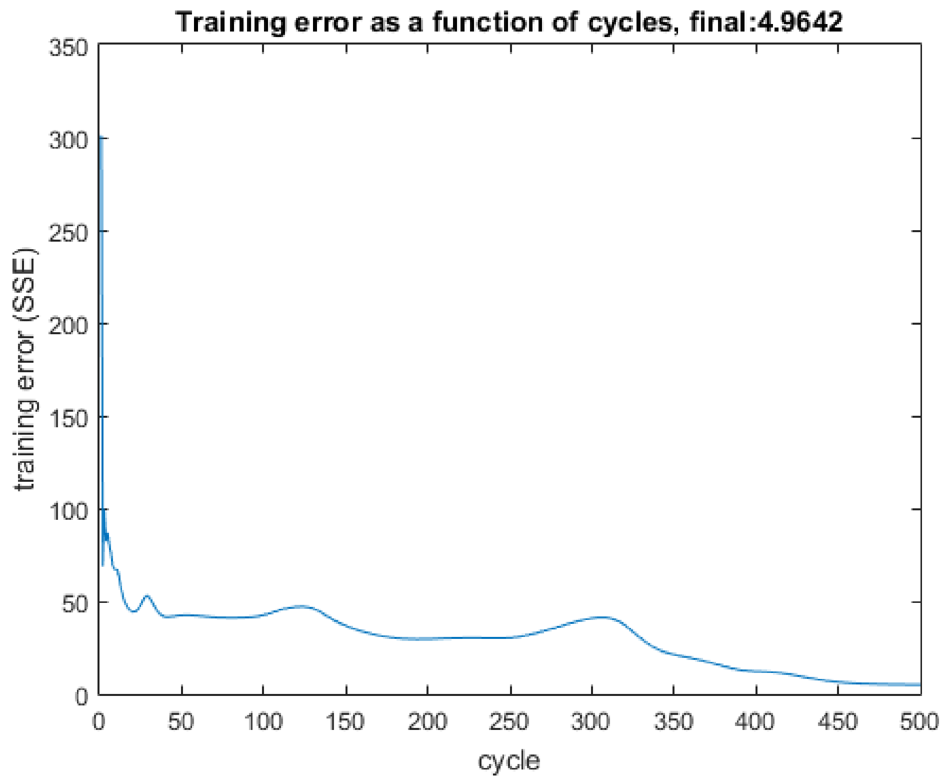


Figure 3: SSE as functions of the number of learning cycle.

```
y_test = myMLP(hUnits,X,Y,cycles,eta);
```

Fig. 6 depicts the development of the SSE over the number of completed learning cycles. To compare the convergence of the MLP on the shuffled data to the training procedure on the unshuffled data, it is instructive to reconsider Fig. 3. As one can observe, and this goes with our expectations, the training procedure converts quicker on the shuffled data set than on the unshuffled data set and takes an overall lower SSE.

We would have expected these results for the following reason: In the sorted data set, points that following each other have correlated values - i.e. neighboring coordinates in the domain take closeby values in the image, as depict in Fig. 1. Thus subsequent weight updates roughly change weights into the same direction - but as the inputs and target values also remain close to each other, the errors get smaller and the magnitude of the gradient vector gets shortened. The result is smaller steps trough the weight space and hence we would expect a slow convergence of the learning procedure. On the other side, when the inputs are processed in randomized order, no such correlation of subsequent error is present and (unless subsequent update steps are always in opposing directions) the weight space will get explored more efficiently.

- Changing the number of hidden nodes: Adding to the number of nodes in the hidden layer allows to learn more complex representations and is always favorable in terms of quality, assessed as the SSE on the training set. However, for an increase in hidden nodes, computation of steps in the learning procedure might also require more CPU time. We found that raising the hidden node number from the default value of 8 to about 50 yields a favorable improvement in regression performance even in light of more run time - but increasing the number drastically, e.g. to 500 does not seem to pay off. Decreasing the number of hidden units is worth considering if we wish to stop the network from learning very

Problem 2 continued on next page. . .

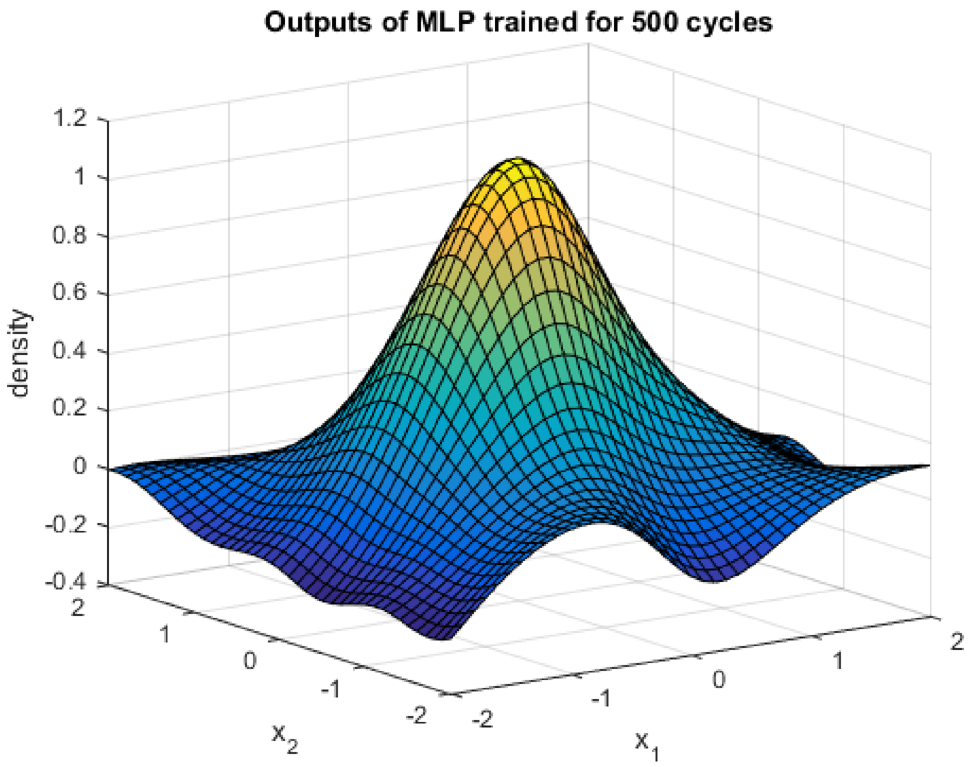


Figure 4: 2D-plot of the trained network output probability density function, after 500 training cycles.

complex representations of the training data (this is a form of regularization), in order to ensure for better generalization of regression performance on previously unseen testing data. As no testing data is provided, this case is no longer investigated here.

- Changing the initial weights: The exploration of the weight space begins an the initial weight parameterization point and depending on this point there might be different local optima in the surrounding neighborhood. Currently, weights are initialized uniformly in the interval $[-0.5, +0.5]$ and we found that different runs of the MLP on differing initial values might in fact lead to differing quality in the goodness of the learned weights. All in all, weights initialized in this range tend to be favorable. Increasing the range to e.g. $[-2.5, +2.5]$ seems to give initial points which are less favorable, as the quality of results decreases and convergence of training seems to take longer - i.e. these points seem to be farther away from a desirable local optimum (or even the global optimum). Furthermore, narrowing the initialization range down from the default values also seems to be detrimental to the quality of the learned results and slows down convergence - it seems that initial weights in the close neighborhood around the zero vector are not a good choice (see exercise (1) on why the zero-vector is a particularly bad choice). Thus we conclude that the current interval $[-0.5, +0.5]$ seems like an already optimized choice. A theoretical argument for this conclusion is the fact that the tanh-nonlinearity also shows its steepest ascent for arguments within this interval, i.e. the entropy over input arguments is particularly high for this interval, and thus the weighted inputs take more distinct values (as opposed to weighted inputs that are either very small or very large and thus fall in a domain where the tanh-nonlinearity saturates, hence all taking closeby values and thus being poorly separable).

- Changing the learning rate: a learning rate is ideal if it lets the learning procedure converge quickly to a point in weight space giving a low SSE. Increasing the default learning rate of 0.1 to larger values

Problem 2 continued on next page. . .

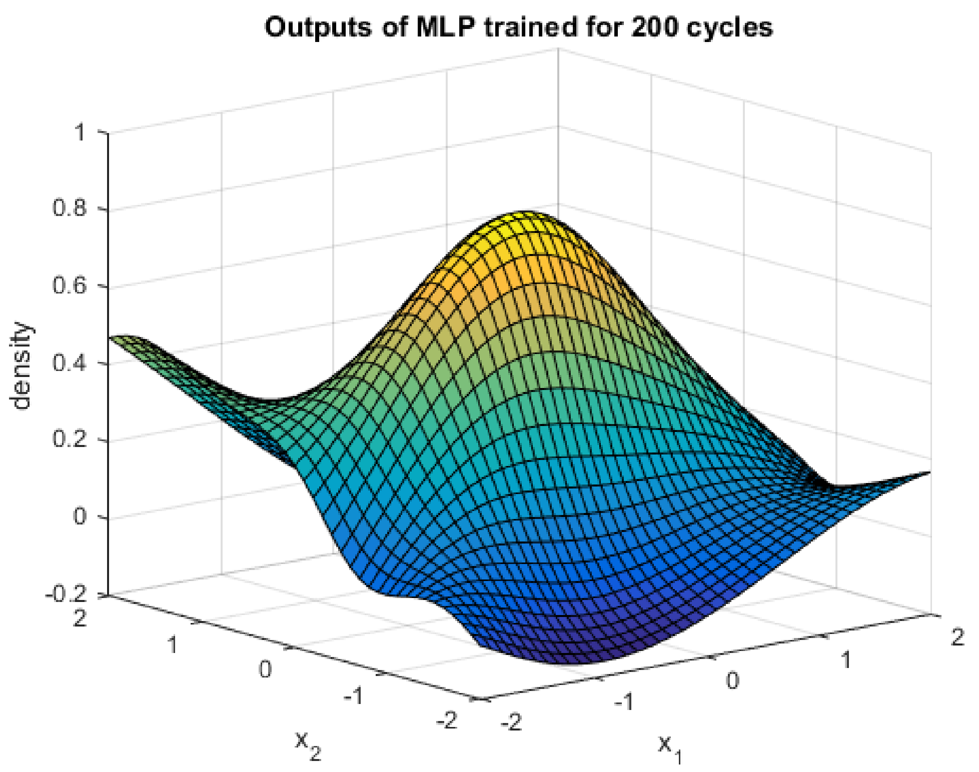


Figure 5: 2D-plot of the trained network output probability density function, after 200 training cycles.

such as 0.5 leads to worse performance. This can be accounted for by the fact that a too large learning rate results in skipping many favorable minima in weight space, i.e. the learning skips over all steep and narrow valleys in the cost function landscape. The other extrema is a too small learning rate, such as e.g. 0.01. This leads to very slow convergence of the learning procedure and is detrimental for the quality of the learned weights, because the learning procedure might stop before it has converged to a good set of weights. Given that neither increasing nor decreasing the learning rate seems to improve the results or the convergence speed, we conclude that a value of 0.1 is already close to optimal...

(5)

A 2D-plot of the target probability density function is shown in Fig. 7 and obtained via the code

```
% load and plot the real data set
data = load('a017_NNpdfGaussMix.txt', '-ASCII');
X = data(:,1:2); Y = data(:,3);

figure();
[x,y] = meshgrid(-2:0.1:2,-2:0.1:2);
surf(x,y,reshape(Y, size(x)))
xlabel('x_1'); ylabel('x_2'); zlabel('density');
title('Multimodal target probability density function')
```

(6)

To train the network on the data set portrait in exercise (5), the network parameters are initialized and the data set is properly randomized via

Problem 2 continued on next page...

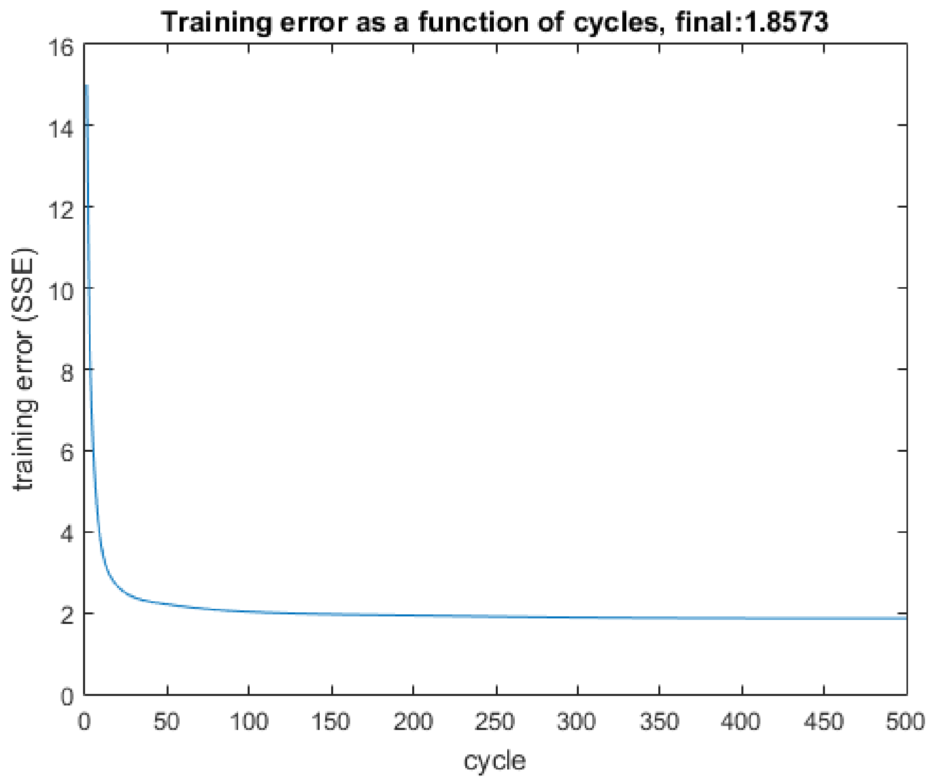


Figure 6: SSE as functions of the number of learning cycle. This is over the shuffled data set as considered in (4)

```
% initialize parameters with which to call the MLP
hUnits = 40;          % specify number of hidden units
eta = 0.01;           % learning rate
cycles = 2000;        % maximal number of cycles over training set

% permute X and Y to a random order
permIdx = randperm(length(X));
X = X(permIdx,:);
Y = Y(permIdx);

% train the MLP on the real data set
y_test = myMLP(hUnits,X,Y,cycles,eta);
```

The results in terms of the development of SSE and the resulting output plot are depict in Figures 8 and 9, respectively.
The final output of the network replicates the coarse shape of the target output probability density function, but it fails to reproduce the finer features, such as the sharp spike around ($x_1 = -0.75, x_2 = 0$). Fig. 10 shows the outputs of a network with 60 hidden units trained for 5000 cycles over the same data set as used above: The plot reveals that the distinct peak that could not be recognized before slowly starts to emergy in this case. An interesting interpretation of this phenomenon is to regard the coarse Gaussian shape of the target distribution as the true target signal, and the sharp spike as some noise. Then we can say that the expanded network starts to model the noise in the data, i.e. a too large network with too many learning steps actually starts overfitting the data.
Even though the Universal Approximation Theorem (Kybenko, 1989) guarantees that a properly parameterized MLP with a sufficient number of hidden units can approximate a given function up to an arbitrary

Problem 2 continued on next page. . .

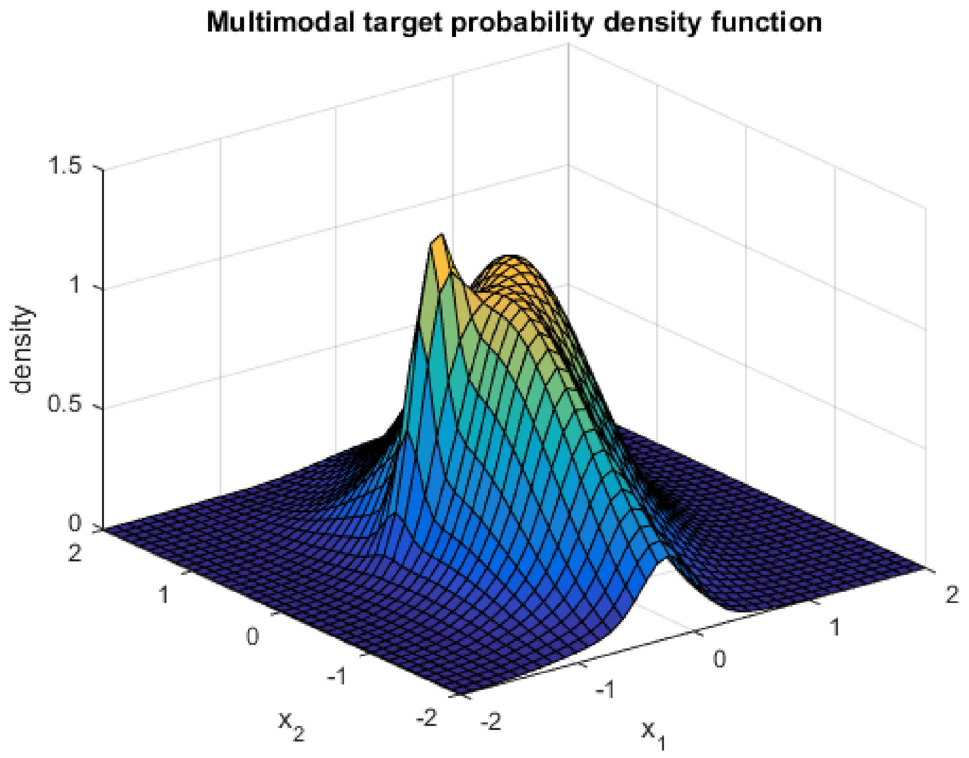


Figure 7: 2D-plot of the target probability density function for the real data set.

precision, having a large enough network with appropriate weights turns out to be very hard in praxis. Besides increasing the number of hidden units (to allow for the learning of more complex feature representations (or noise?), see e.g. 10), a straightforward attempt to learn better weights is to decrease the learning rate and to increase the number of training cycles. The increase of cycles and the decrease of the learning rate might avoid to step over the global optimum (or favorable local optima), but could result in getting trapped in unfavorable local optima (that is: a weight parameterization that is the optimum within its local neighborhood, but not getting close to the global optimum weight). To guard against this issue, one can incorporate a momentum term into the update step of the weights

```
%% [back propagate] errors and calculate derivatives
d2 = error;                         % back propagate to hidden layer, eq. (5.65)
d1 = (1-z.^2)' .* (d2*w2);          % back propagate to input layer, eq. (5.66)

dE2 = d2*z;                         % get error derivative, Bishop eq (5.53),
dE1 = d1*X(seq,:);                  % analogous to Bishop eq (5.67)

dw2 = eta * dE2 + mom * dw2;        % update weights, for mom = 0 this is
dw1 = eta * dE1 + mom * dw1;        % equivalent to Bishop eq (5.41)
```

The function of the momentum term is to give the gradient descent procedure a drift property (think of an inertial mass that continues to move in its original direction) such that the gradient goes, by a small scalar factor *mom*, in the direction of the previous step. Thus, even when we reach a point where the gradient vanishes due to the parameterization hitting a local minimum, one might hope to slide over the local minimum thanks to the additive drift term. An appropriately chosen value of *mom* could result in increased convergence speed and better end results of the learning procedure.
In addition, one could instantiate multiple networks with different initial weights and, after the set of net-

Problem 2 continued on next page. . .

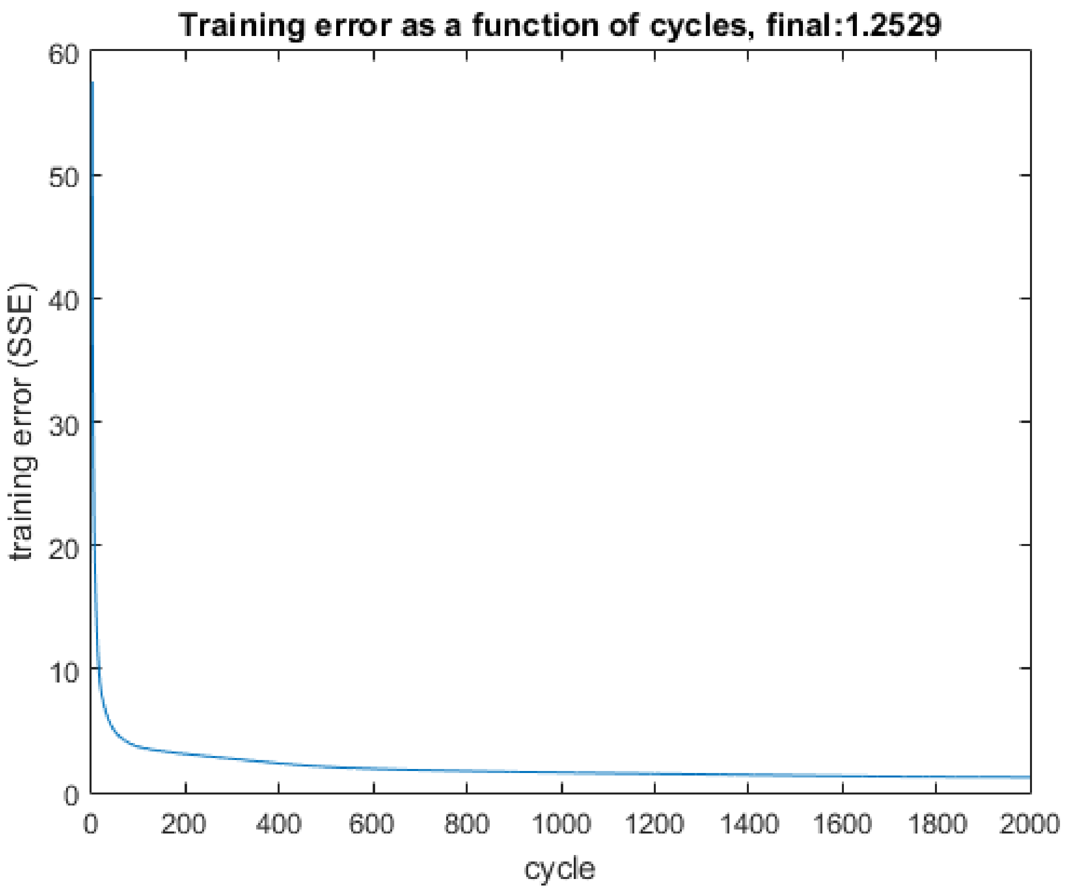


Figure 8: SSE as function of the number of learning cycle.

works has completed learning, choose the one with the best parameters. The idea behind the multiple instances is to explore the weight space from different starting points, corresponding to the networks' initial weights. While some instances might get trapped in local optima of the cost function, other networks might converge to the global optimum or at least a more favorable local optimum.

(7)

An MLP is created and trained with the aid of the netlab toolbox via the code

```
% initialize parameters with which to call the MLP
NIN = size(X,2);     % define dimension of input
hUnits = 40;         % specify number of hidden units
NOUT= size(Y,2);     % define dimension of output
FUNC='linear';       % define output function

% instantiate the network
NET = mlp(NIN, hUnits, NOUT, FUNC);

ALG = 'scg';         % define training mode: Conjugate Gradient Descent
OPTIONS=foptions;    % set options parameters to default values
OPTIONS(14) = 2000; % except for number of learning cycles, set to 2000

% learn the network weights
[NET, OPTIONS] = netopt(NET, OPTIONS, X, Y, ALG);
```

Problem 2 continued on next page...

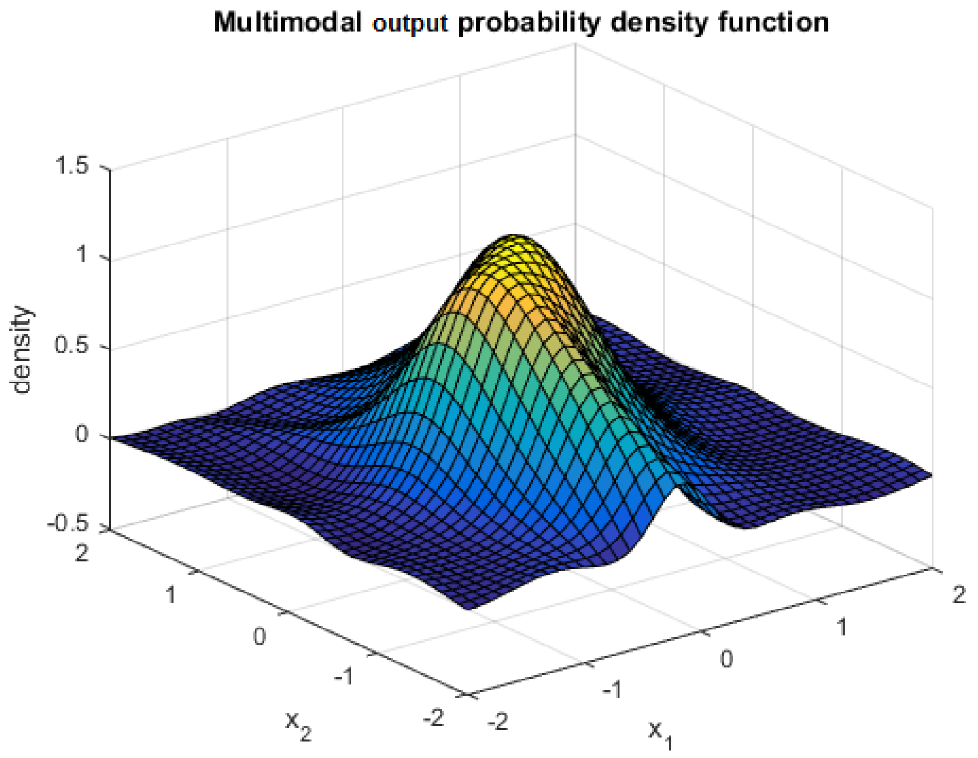


Figure 9: 2D-plot of the trained network output probability density function for the real data set.

```

% forward propagate X through the learned NET to get predictions
predictions = mlpfwd(NET, X);
```

To evaluate the goodness of the regression performance of the learned network we compute the sum-of-squares error on the data set

```
% assess goodness of predictions via SSE
SSE_netlab = 0.5 * sum((Y-predictions).^2);

>> SSE_netlab

SSE_netlab =

    1.9714
```

It is important to notice that this performance is by a network trained with scaled conjugate gradients, where a gradient descent is performed in weight space in directions conjugate[1] to each other. Scaled conjugate gradients makes also use of the error backpropagation technique, but it explores state space in a more efficient way than our training method - thus both methods are hard to compare e.g. in terms of the employed learning rate or the number of cycles over the training set being used. However, we used 40 hidden units for both training types and set the number of complete cycles over the data set to 2000 for both methods. This word of caution being said, we notice that the scaled conjugate gradient training runs way faster than our method, but yields a worse SSE on the training set (as compared to the SSE reported in Fig. 8). This might be because the netlab network might get regularized (but we didn't find any informations on this in

[1]Conjugacy is related to orthogonality. The interested reader is referred to 'An Introduction to the Conjugate Gradient Method Without the Agonizing Pain' (Shewchuk, 1994) for a splendid introduction to the topic.

Problem 2 continued on next page. . .

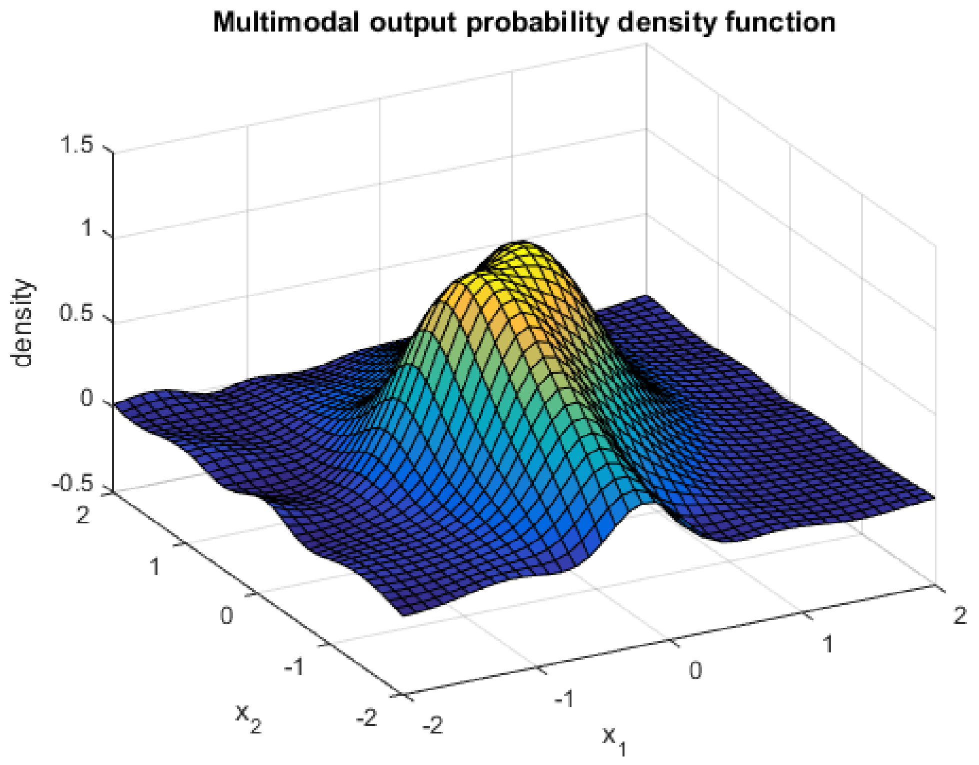


Figure 10: 2D-plot of the trained network output probability density function for the real data set, 5000 cycles and 60 hidden units. We can observe that the bump gets more emphasized as compared to Fig. 9.

the documentation or the help of the function foptions, so we don't know how to turn it off): if this would be the case, then the network would be stopped from learning a perfect representation of the input data, including the noise (which is known as 'overfitting'), in order to allow for better generalization on validation or testing data - which is not considered here however. So to allow for a final conclusion, we would require to run both networks on a previously unseen testing data set and compare both performances in terms of SSE on the new data set.

Problem 2 continued on next page. . .

Problem 3

In this exercise, the EM algorithm is applied to a blood test sample to identify whether a tester has taken drug X or not. 2000 samples were collected and four different values measured. The test sample results therefore into a 2000×4-dimensional set. It is suggested that 20% of the population might take drug X.
A new discovery has shown that there is a high correlation between x_1 and x_2 if drug X is present in the blood.
A mixture of K-Gaussian is assumed:

$$p(\mathbf{x} \mid \mu, \mathbf{\Sigma}, \pi) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x} \mid \mu_k, \mathbf{\Sigma}_k). \tag{1}$$

where $\mathbf{x} = [\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4]$ are the measures of the four different quantities measured in the blood sample and $\mu = \{\mu_1, \dots, \mu_K\}$ and $\mathbf{\Sigma} = \{\Sigma_1, \dots, \Sigma_K\}$ are the means and covariates of the Gaussian distributions for each class $k = \{1, \dots, K\}$. The mixture weights are represented in $\pi = \{\pi_1, \dots, \pi_K\}$ with $\sum_{k=1}^{K} \pi_k = 1$.

Exploratory Data Analysis

To guess an appropriate value for K, π and μ and $\mathbf{\Sigma}$, histograms and scatter plots are created (ref Fig 11) and figures in subsection .

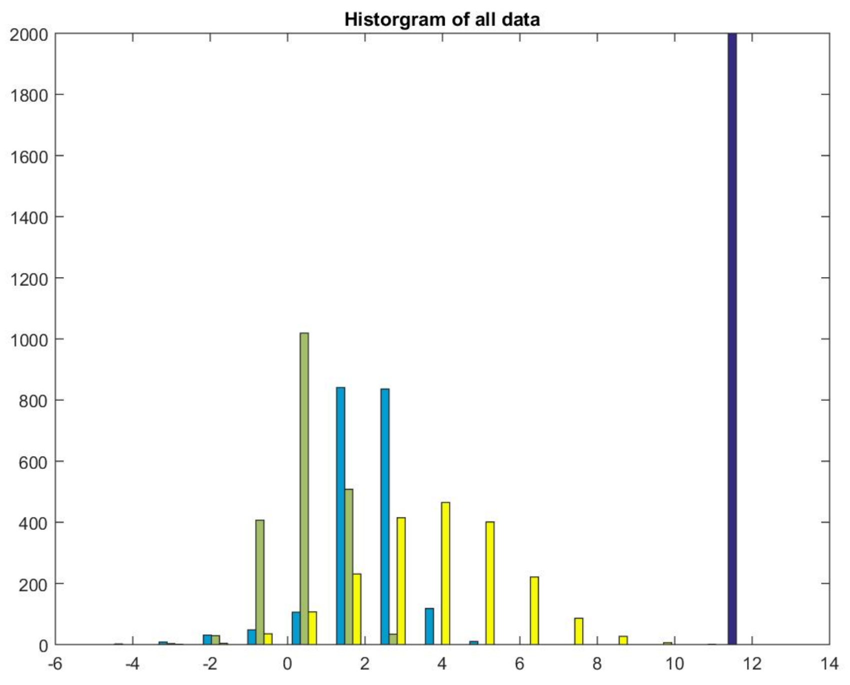


Figure 11: Scatter-plot of Test and Training Data to function

Based on the scatter plots of only two variables at each time, one can see that $[x_3, x_4]$ and $[x_1, x_4]$ are not really correlated (see figures 20 - 25). The others seem to be correlated in various manners. That would leave four remaining correlations between each two variables. In the histogram of all data (fig 11), one can see that two variables (blue and green) have values closer together whilst one (dark-blue) seems to be independent with values around 12.0. The third (yellow) distributes in a nice bell-shape that is typical for Gaussian distributions in-between the area of the first two mentioned (blue and green) and the last one (dark-blue). Therefore, we'd suggest a $K = 3$ or $K = 4$-mixture model with higher preference on a $K = 4$-mixture.

Implementation of EM

The EM-algorithm consists of four steps, following Bischop Eq: 9.23-9.28:

1. initialize the parameters $\mu_0, \mathbf{\Sigma}_0, \pi_0$:

$$\begin{aligned} \mu_0 &= \bar{x} + [-1 \leq \epsilon \leq +1] && (2) \\ \mathbf{\Sigma}_0 &= diag\,(4 \times rand() + 2) && (3) \\ \pi_0 &= \frac{1}{K} && (4) \\ & && (5) \end{aligned}$$

   where $\epsilon = rand() * 2 - 1$, a random variable in the interval $[-1, 1]$, $\bar{x}$ is the empirical mean for each variable $[x_1, x_2, x_3, x_4]$. $\mathbf{\Sigma}_0$ is the diagonal matrix of random variables of the interval $[2, 6]$ and the weights are equally distributed.

2. E-step calculate z_{nk}, the responsibility for each element n to belong to class k.

$$\gamma\,(z_{nk}) = \frac{\pi_k \mathcal{N}\,(\mathbf{x_n} \mid \mu_\mathbf{k}, \mathbf{\Sigma}_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}\,(\mathbf{x_n} \mid \mu_\mathbf{j}, \mathbf{\Sigma}_j)} \qquad (6)$$

3. M-step recalculate the parameters by maximizing equation 6:

$$\begin{aligned} \mu_{new} &= \frac{1}{N_k} \sum_{n=1}^{N} \gamma\,(z_{nk})\, \mathbf{x}_n && (7) \\ \mathbf{\Sigma}_{new} &= \frac{1}{N_k} \sum_{n=1}^{N} \gamma\,(z_{nk})\,(\mathbf{x}_n - \mu_k^{new})\,(\mathbf{x}_n - \mu_k^{new})^T && (8) \\ \pi_{new} &= \frac{N_K}{N} && (9) \\ & && (10) \end{aligned}$$

   with $N_K = \sum_{n=1}^{N} \gamma\,(z_{nk})$.

4. compute the likelihood:

$$\ln p\,(\mathbf{X} \mid \mu, \mathbf{\Sigma}, \pi) = \sum_{n=1}^{N} \ln \left(\sum_{k=1}^{K} \pi_k \mathcal{N}\,(\mathbf{x_n} \mid \mu_\mathbf{k}, \mathbf{\Sigma_k}) \right), \qquad (11)$$

   compare it to its previous value and in case the difference is bigger than a certain threshold δ go back to step 2.

This algorithm is implemented in `Assignment4_Ex3.m` and `EM.m` in the belonging folder of `Ex3`. In each iteration, the log-likelihood development is depicted (see Fig 12)and the variables $\mathbf{x_1}, \mathbf{x_2}$ are shown by coordinates in the colors of the class they most probably belong to according to the mixture model (see fig 13). The depiction is created by `vizClass.m`.

Problem 3 continued on next page. . .

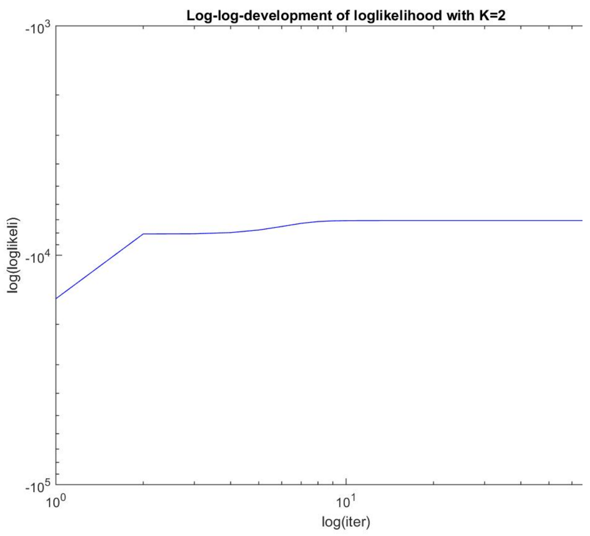


Figure 12: Log-likelihood assuming $K = 2$ clusters

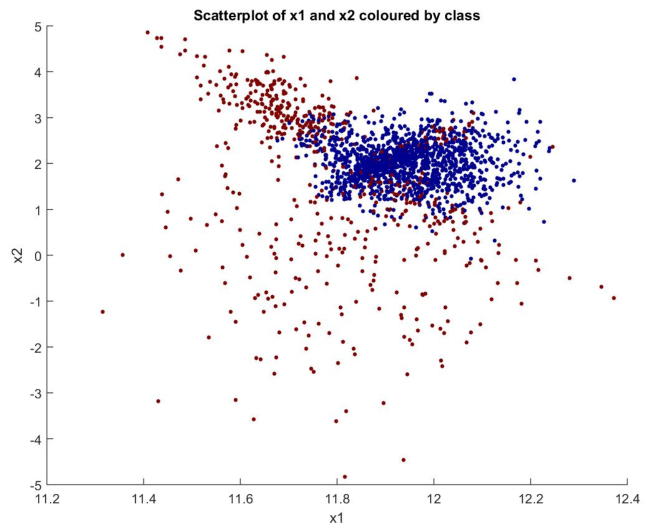


Figure 13: Scatter-plot assuming $K = 2$ clusters

Apply EM to data with $K = 2$

The EM algorithm written in `EM.m` is applied to the data with two classes assumed ($K = 2$). The development of the log-likelihood can be seen in figure 12 and the final categorization of data $\mathbf{x}_1, \mathbf{x_2}$ is depicted in figure 13.
Despite random initializations of the Gaussian variables $\mu_\mathbf{0}$ and $\mathbf{\Sigma_0}$, the EM algorithm converges quickly. In the first few steps of the algorithm, a big difference in log-likelihood values (evaluated after each step of the EM algorithm) can be observed (by mostly halving its value). Then the values converge quickly (comp. figure 12). No strong correlation within the clusters can be seen by eye in figure 13.
For $\mathbf{x}_1, \mathbf{x_2}$ a strong positive correlation is assumed in case drug $\mathbf{X}$ is present. To verify that, the correlation coefficients according to

$$\rho_{12,k} = \frac{cov\left[\mathbf{x}_1, \mathbf{x_2}\right]}{\sqrt{var\left[\mathbf{x}_1\right] var\left[\mathbf{x_2}\right]}}, \tag{12}$$

are calculated for each cluster $k = \{1, \ldots, K\}$. The function `isStrong.m` verifies whether any $\rho_{12,k}$ is larger than 0.5 to confirm a strong positive correlation, that is assumed between $\mathbf{x_1}$ and $\mathbf{x_2}$. No strong correlation within the clusters can be seen in figure 13, and no strong correlation between x_1 and x_2 is found numerically.

EM with $K = 3$ and $K = 4$

Rerunning the EM algorithm with the same initialization as before (but adapted to K=3), results in the development of the log-likelihood depicted on the left in fig 14 (depicted as a log-log-plot) and a new classification of the variables $\mathbf{x}_1, \mathbf{x_2}$ depicted on the right-hand side.
Evaluating the correlation coefficients according to equation 12, the first group (depicted in yellow), has a strong negative correlation coefficient with $\rho_{12,K=3} = -0.7229$. But none of the three groups show a strong positive correlation.
The results of running the EM-algorithm with $K = 4$ clusters assumed are depicted in figure15. One can see a strong positive and a negative correlation of the clusters depicted in purple and yellow, respectively.
Now, as expected, the program `isStrong.m` indicates that the classes $k = 1$ and $k = 3$ have strong correlation coefficients of -0.8944 and 0.9154 respectively. As a strong positive correlation between $\mathbf{x_1}$ and $\mathbf{x_2}$ is suggested to identify the drug, all data belonging to group 3 would then identify the presence of drug X.

Problem 3 continued on next page...

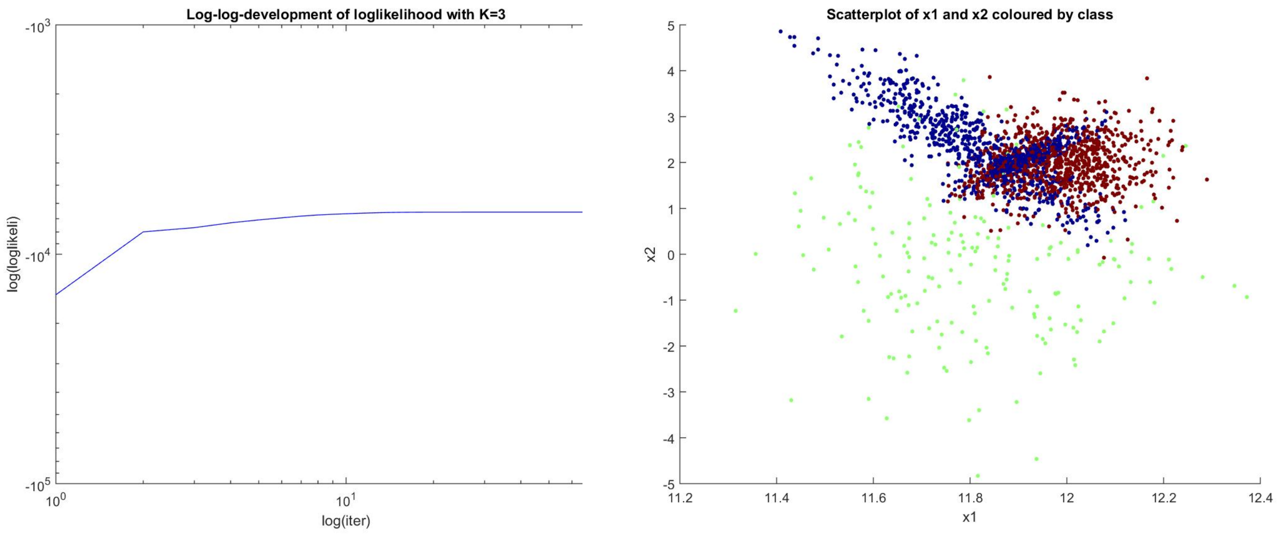


Figure 14: Log-likelihood and scatter-plot of $\mathbf{x_1}$ and $\mathbf{x_2}$ colored by cluster belonging with 3 clusters assumed.

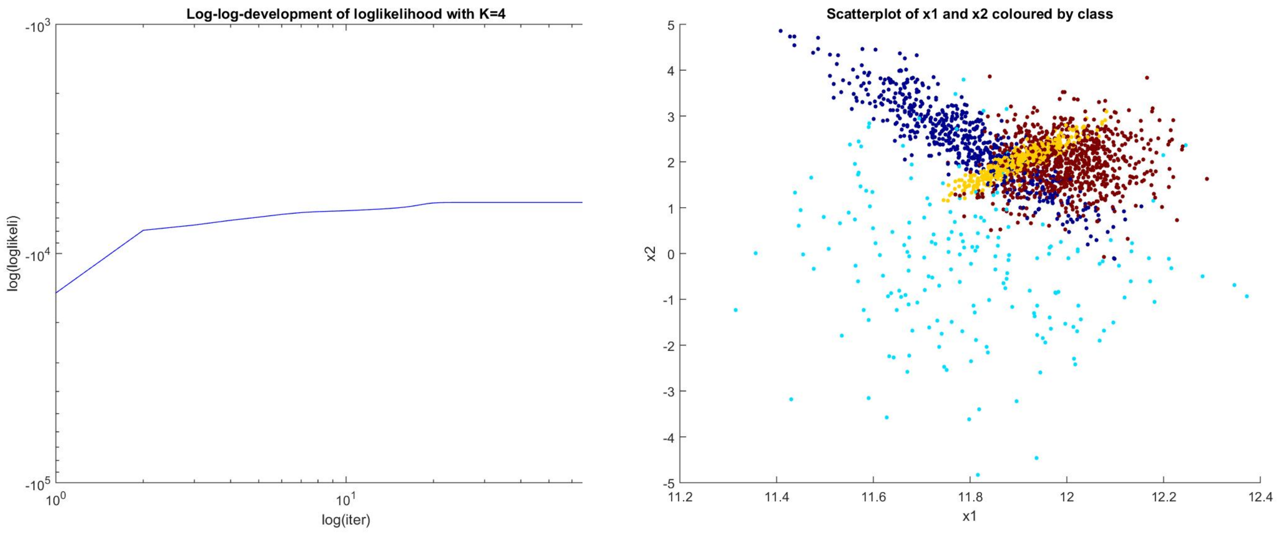


Figure 15: Log-likelihood and scatter-plot of $\mathbf{x_1}$ and $\mathbf{x_2}$ colored by cluster belonging with 4 clusters assumed.

The estimated weight for the mixture model of $K = 4$ groups is:

$$\pi_4 = \{0.2930, 0.1064, 0.1958, 0.4048\}.$$

This proves the rumors of every fifth person to consume drug X right.

Problem 3 continued on next page...

Identify the "tamper" and the "fraud

Classifying the four new probes,

$$\begin{aligned} A &= [11.85, 2.2, 0.5, 4.0] && (13) \\ B &= [11.95, 3.1, 0.0, 1.0] && (14) \\ C &= [12.00, 2.5, 0.0, 2.0] && (15) \\ D &= [12.00, 3.0, 1.0, 6.3] && (16) \end{aligned}$$

into the four estimated classes, results in the classification into the groups $4, 1, 3, 4$, respectively. Therefore, subject C is tested positively on drug X. As class $k = 1$ showed high negative correlation between x_1 and x_2, subject B can be identified having tampered with the test by artifically altering some x_i levels.

Attachement to Exploratory Data Analysis

Problem 3 continued on next page. . .

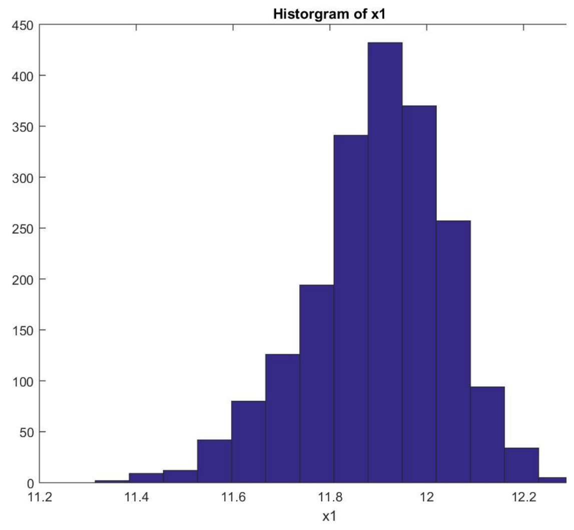


Figure 16: Histogram of x_1

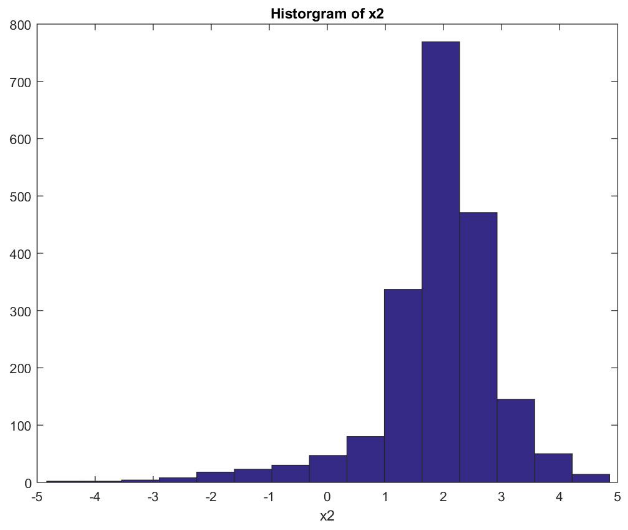


Figure 17: Histogram of x_2

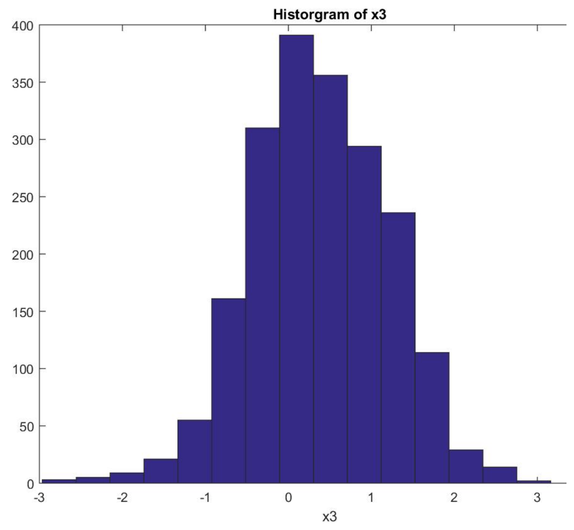


Figure 18: Histogram of x_3

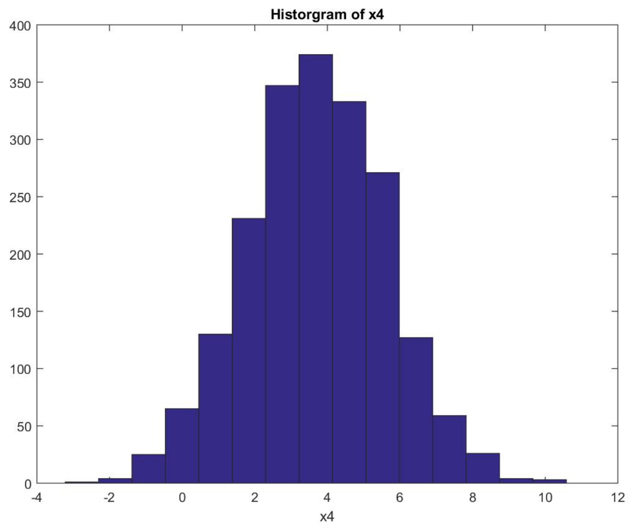


Figure 19: Histogram of x_4

Problem 3 continued on next page. . .

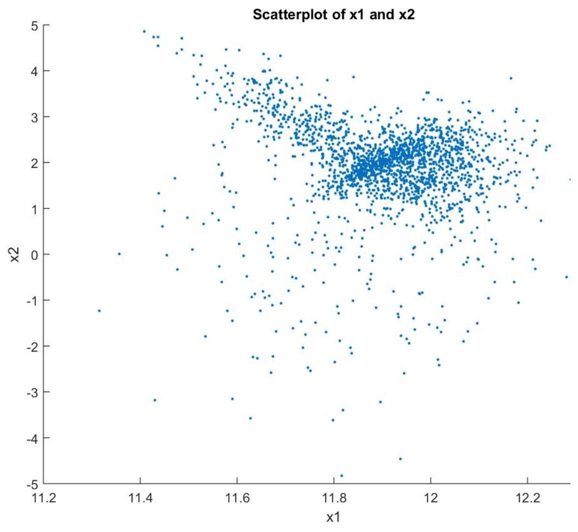


Figure 20: Scatter plot of x_1 and x_2

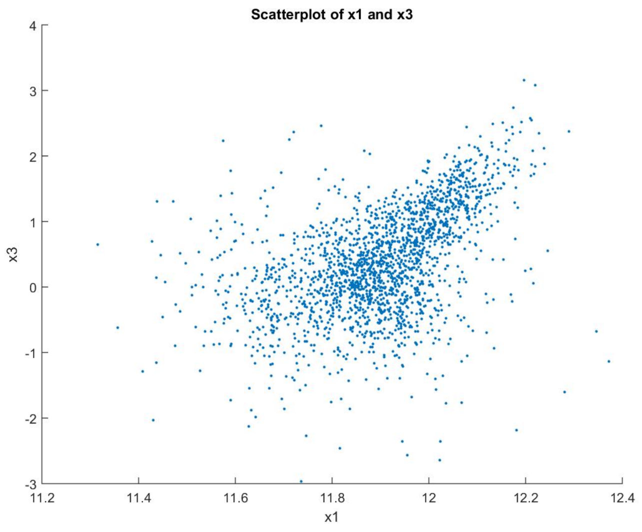


Figure 21: Scatter plot of x_1 and x_3

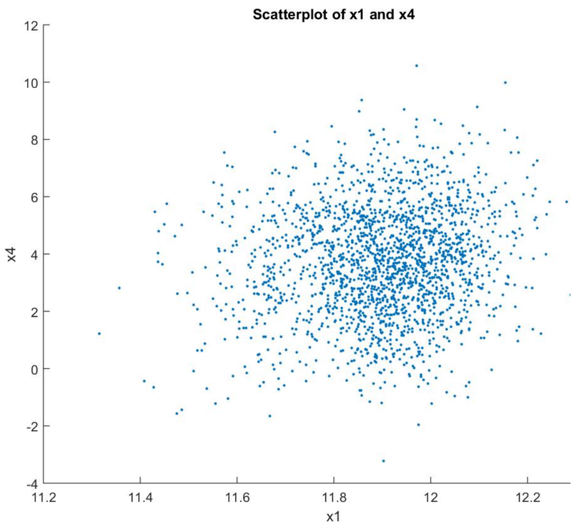


Figure 22: Scatter plot of x_1 and x_4

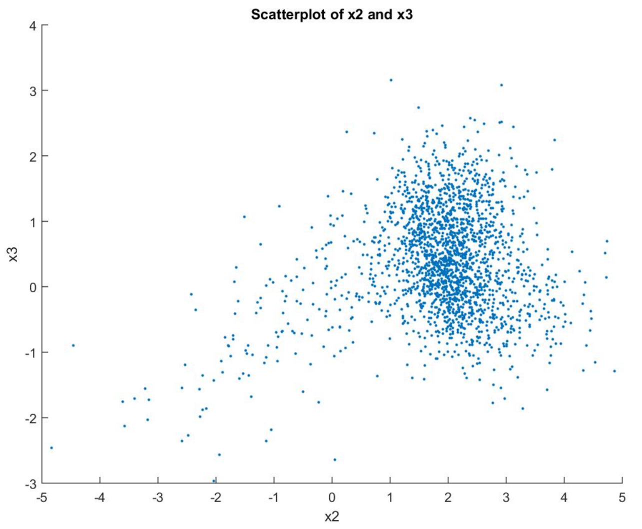
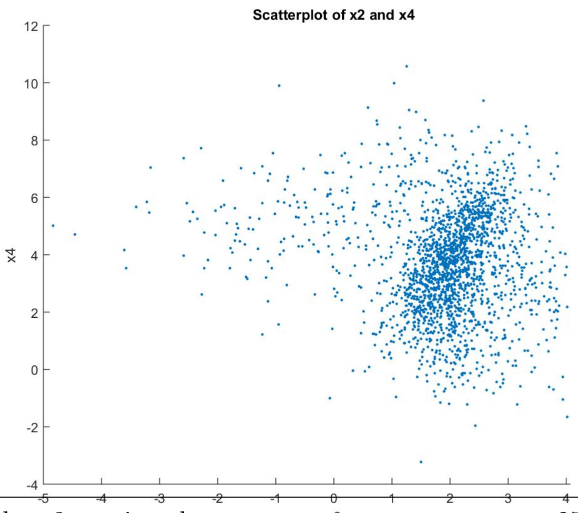
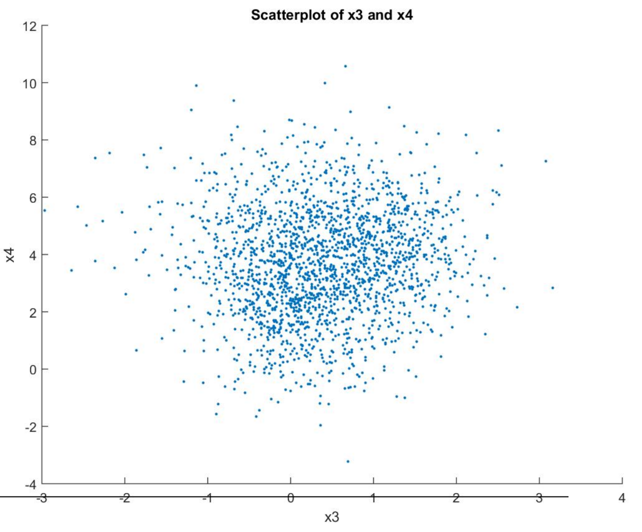


Figure 23: Scatter plot of x_2 and x_3





Problem 3 continued on next page. . .

Figure 24: Scatter plot of x_2 and x_4

Figure 25: Scatter plot of x_3 and x_4

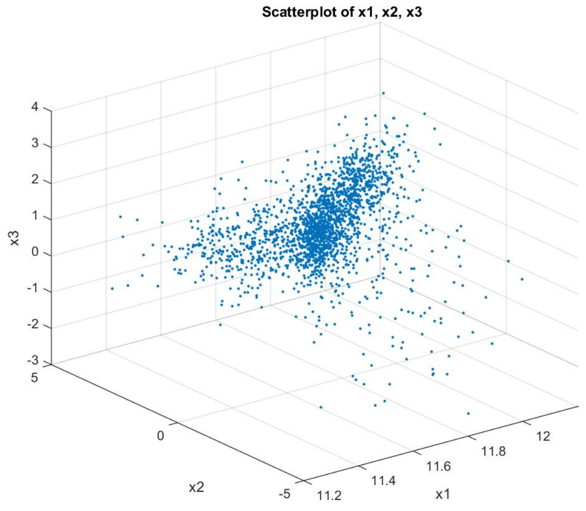


Figure 26: Scatter plot of x_1, x_2, x_3

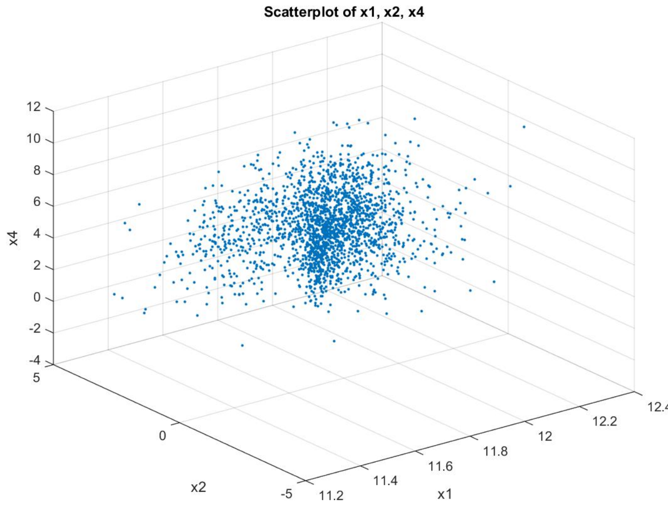


Figure 27: Scatter plot of x_1, x_2, x_4

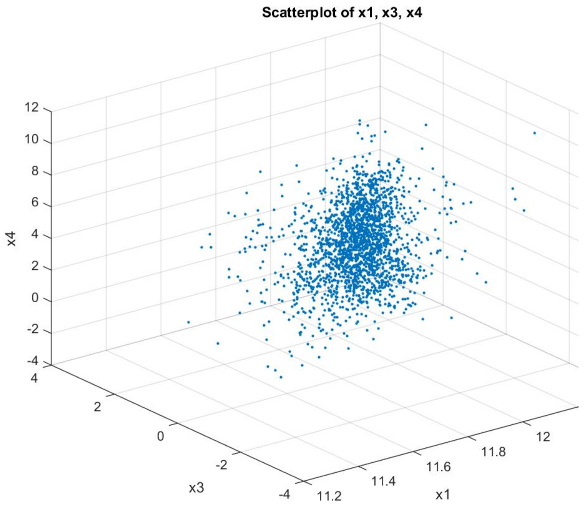


Figure 28: Scatter plot of x_1, x_3, x_4

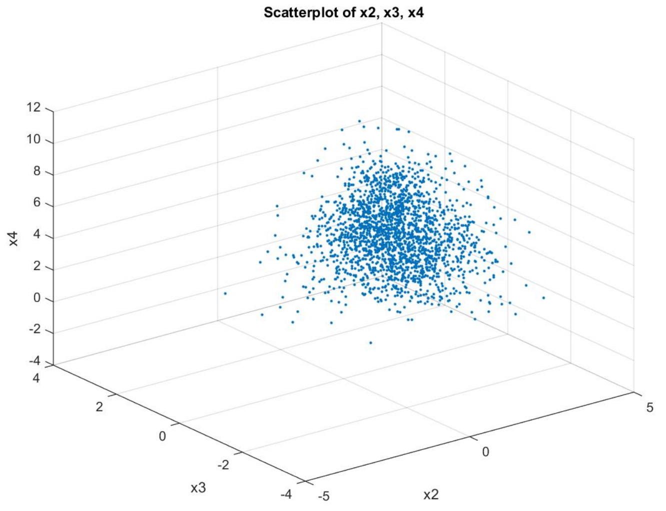


Figure 29: Scatter plot of x_2, x_3, x_4

Problem 3 continued on next page...

Problem 4

Now, we apply the EM algorithm to recognize handwritten digits of the number $'2'$, $'3'$ and $'4'$. On the basis of 800 different handwritten digits, each of 28×28 pixels, either white or black. Due to the binomial nature of the data $(xi, d \in \{0, 1\}$, A Bernoulli mixture model is trained. The data is stored in $X \in [0, 1]^{800\times 28*28}$. Each image $\mathbf{x}_i$ is stored in row i of $\mathbf{X}$.
A mixture of K-Bernoulli is assumed:

$$p(\mathbf{x} \mid \mu, \pi) = \sum_{k=1}^{K} \pi_k \prod_{i=1}^{D} \mu_{ki}^{x_i} (1 - \mu_{ki})^{(1-x_i)} . \tag{17}$$

where μ_{ki} is the probability that pixel i in class k is black and $\pi = \{\pi_k, \dots, \pi_k\}$ are the mixing coefficients on the overall dataset with $\pi = \sum_{k=1}^{K} \pi_k = 1$.
The aim of the exercise is to classify new images of handwritten numbers of $'2'$, $'3'$ and $'4'$.

EDA

To get an impression of the data $\mathbf{X}$, some randomly chosen pictures are visualized with `image()` (cp. figure 30). The row numbers i are randomly chosen and each x_i is first transformed into a 28×28 matrix and then displayed with a multiplying factor of 100 for the function `image()` to pick up the difference of $x_{i,j} = 0$ or $x_{i,j} = 1$.
The figures are mostly clearly distinguishable and the numbers recognizable. Image 102, on the left second row of fig 30, might be difficult to categorize even though it has the typical bottom shape of a $'3'$. Generally we expect a good portion of the images to be identified correctly. Mal-written digits will be more difficult to distinguish.

Implementation of EM for Bernoulli-mixture model with $K = 3$

The EM-algorithm for the Bernoulli-mixture model is implemented in `EM_Bernoulli()`. The EM-algorithm is now of the following form, Bischop eq. (9.23 - 9.28):

1. The initial values are given as the following:

$$\mu_{\mathbf{0}} = 0.25 + rand(K, 28 * 28) * 0.5 \tag{18}$$
$$\pi_0 = \frac{1}{K}, \tag{19}$$

   where $\mu_{\mathbf{k}}$ takes random values between $[0.25, 0.75]$ and π are assumed to be equal for each class.

2. E-step calculate z_{nk}, the responsibility for each element n to belong to class k.

$$\gamma(z_{nk}) = \frac{\pi_k \prod_{i=1}^{D} \mu_{ki}^{x_i} (1 - \mu_{ki})^{(1-x_i)}}{\sum_{j=1}^{K} \pi_k \prod_{i=1}^{D} \mu_{ki}^{x_i} (1 - \mu_{ki})^{(1-x_i)}} \tag{20}$$

3. M-step recalculate the parameters by maximizing 20:

$$\mu_{new} = \frac{1}{N_k} \sum_{n=1}^{N} \gamma(z_{nk}) \mathbf{x}_n \tag{21}$$
$$\pi_{new} = \frac{N_K}{N} \tag{22}$$
$$\tag{23}$$

   with $N_K = \sum_{n=1}^{N} \gamma(z_{nk})$.

Problem 4 continued on next page. . .

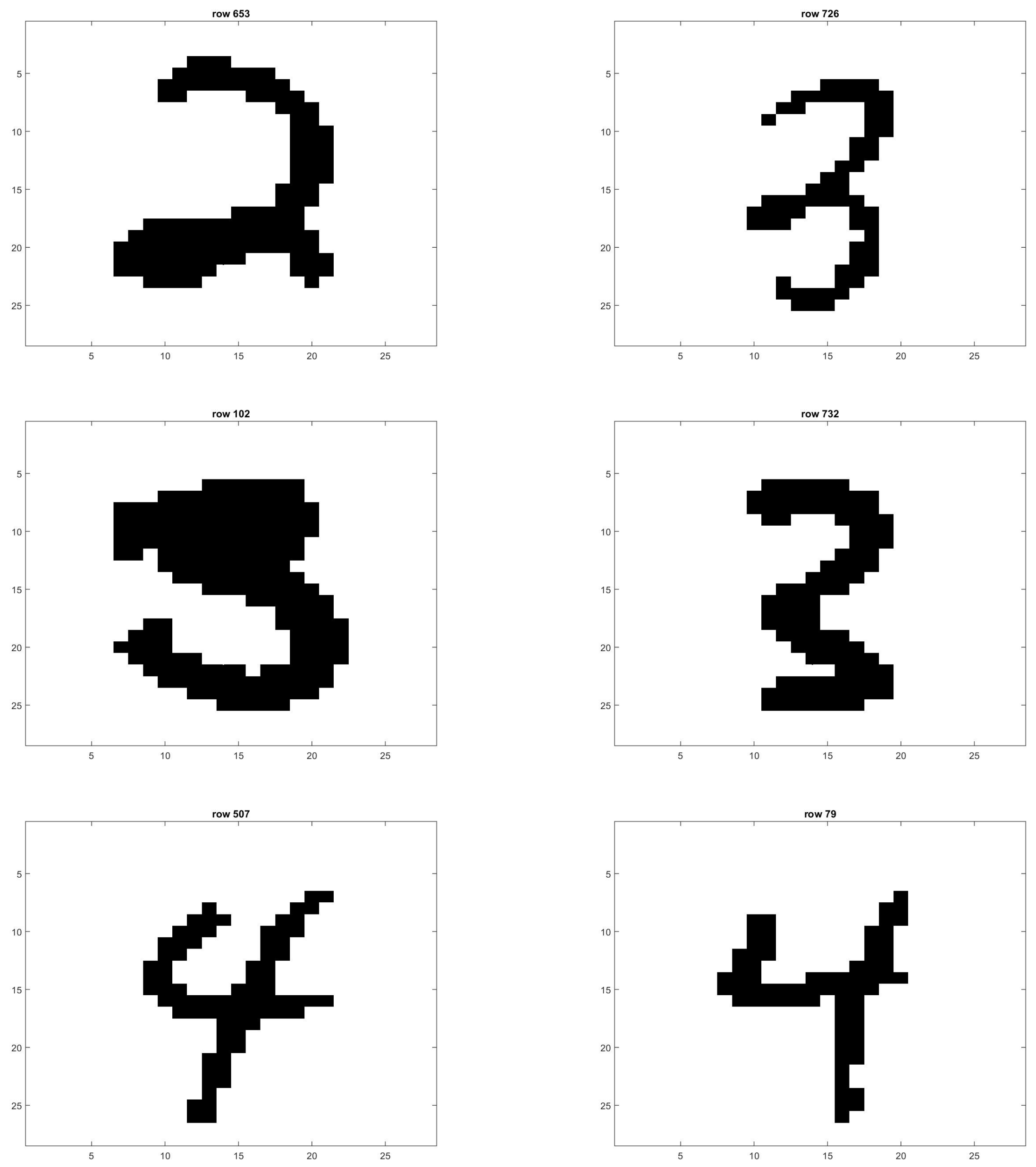


Figure 30: Six randomly chosen digits, two of each category $'2','3'$ and $'4"$.

As the likelihood terms per class, $\pi_k \prod_{i=1}^{D} \mu_{ki}^{x_i} (1-\mu_{ki})^{(1-x_i)}$ become very small, the logarithm of the likelihood is first calculated and then translated back with `exp()`. Therefore, one gets:

$$\pi_k p(\mathbf{x}_n \mid \mu_{\mathbf{k}}) = \exp\left\{\log\left[\pi_k \prod_{i=1}^{D} \mu_{ki}^{x_i} (1-\mu_{ki})^{(1-x_i)}\right]\right\} \tag{24}$$

$$= \exp\left\{\log(\pi) + \sum_{i=1}^{D} \left[x_i \log(\mu_{ki}) + (1-x_i)\log(1-\mu_{ki})\right]\right\}. \tag{25}$$

This numerical trick allows to calculate the small probabilities without them being numerically rounded to 0 too soon.

Problem 4 continued on next page. . .

This algorithm is implemented in `Assignment4_Ex4.m` and `EM_Bernoulli.m` in the belonging folder of `Ex4`. Further, in each step, the pictures based on the current $\mu_{\mathbf{k}}$ are depicted (see fig 31).

EM run on K=3

Again, the effect of different randomization is not noticeable. The clusters get quickly a clearer shape of the digit their representing. The resulting estimation for the three classes can be seen in fig 31.

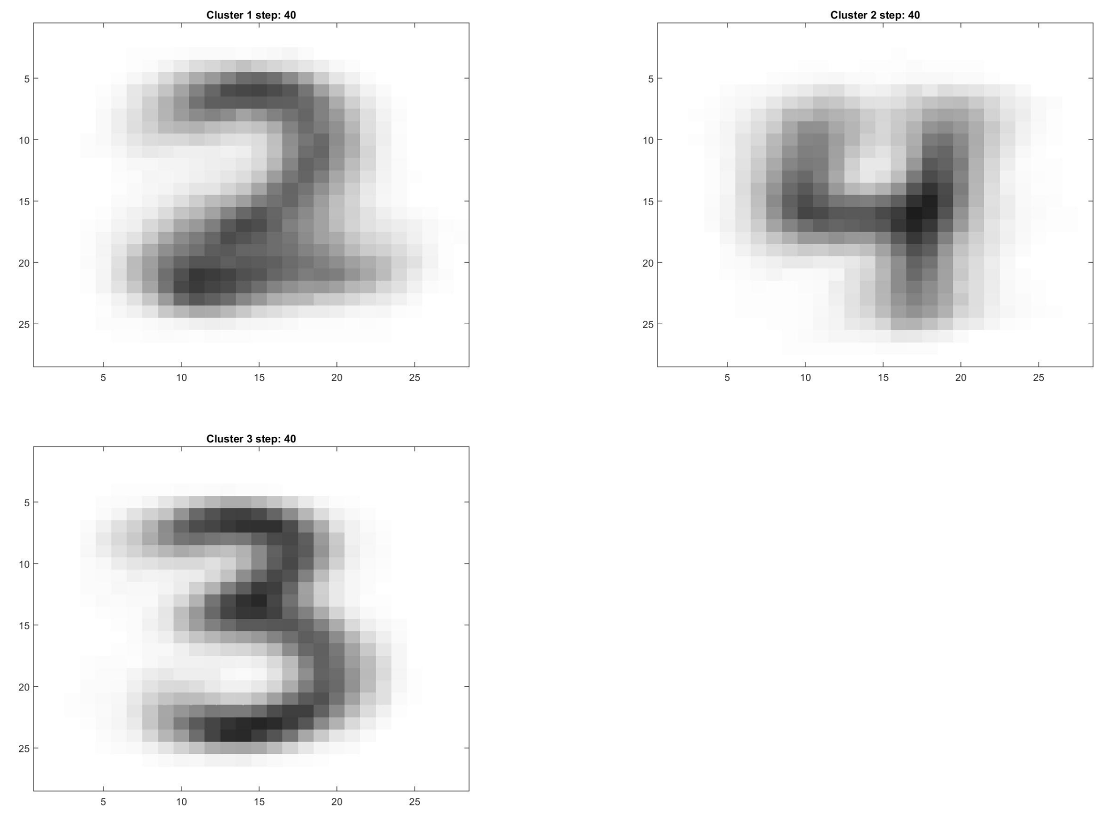


Figure 31: Clusters estimated with EM() for $K = 3$

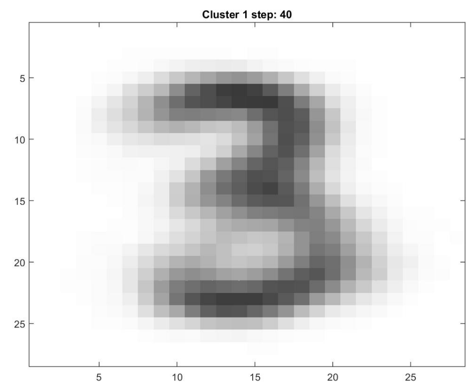
One can see, that the clusters $k = \{1, 2, 3\}$ model the digits $'2', '4', '3'$ respectively. The digits are each clearly identifiable. Further, the darkest spots represent digits that strongly indicate these pixels belonging to the given class. For example, the bottom of the digit $'3'$, as earlier stated being unique for the shape of the digit, is considered more important for this digit.

EM on $K = 2$ and $K = 4$, compared to true labels and run with true initial values on $K = 3$

EM on $K = 2$ and $K = 4$

The algorithm is also run on $K = 2$ and $K = 4$. This clearly doesn't solve the problem of identifying the digits. For $K = 2$, although the digit $'3'$ might be identifiable, the other cluster is not (cp. fig. 32). One gets the impression, the digit $'2'$ is modeled in both classes. For $K = 4$ the probabilities become so small, that despite the numerical trick applied, they are rounded to 0 and no useful result is obtained.

Problem 4 continued on next page...



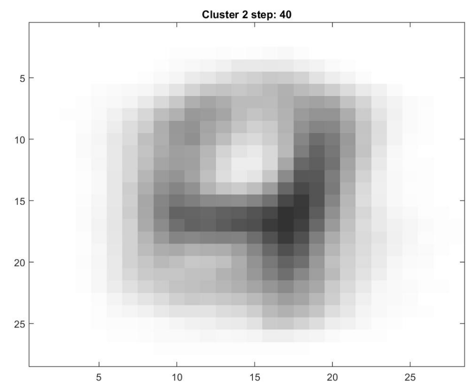


Figure 32: Clusters estimated with EM() for $K = 3$

Compare true labels

The estimated labels are compared to its true values. The number of misclassified digits for $'2','3','4'$ are $12, 27, 9$' respectively, 48 in total. The estimated weights are $\pi_{K=3} = [0.31487, 0.37627, 0.30886]$. Whilst for the true π_X holds $\pi_X = [0.3, 0.4, 0.3]$. These are quite good estimates taking into account the whole set of images were used for the optimization. As stated already when having a first look at the data (see fig. 30, a few digits might be difficult to classify as they're already not easily classifiable by eye. Some misclassified digits are depicted in figures 33. In the examples the $'2'$ where classified into class $'3'$, $'3'$ into cluster of $'4'$ and $'4'$ into the cluster of $'2'$. This is just a random coincidence: the misclassified digits are classified into both of the other two clusters.
A possibility to improve the algorithm would be to go through the picture sample $\mathbf{X}$ and exclude pictures that are not immediately or only with difficulty identifiable by eye.

Run with true labels

Running the algorithm with true labels still leads to a misclassification of 48 pictures (the same as with the random initialization), although the initial values are not really different from the values estimated with the EM algorithm. In figure 34, on the left-hand side, the classifiers are depicted that were estimated with the EM-algorithm with the initial values set to the true ones. On the right-hand side the true classifiers are depicted that results from the labels from the true data . There are almost no differences between the estimated cluster-pattern and the actual cluster pattern.

Analyze own handwritten digit

In the last step of the analysis of handwritten digits, we created an own handwritten image of the digit $'2'$ (see fig 35) The probability for the handwritten image to belong to each class is calculated following equation 20. For the values of μ and π, the calculated values from the output of the EM-algorithm with $K = 3$ are used. The digit is perfectly categorized into cluster 1, the cluster representing the digits $'2'$.

Problem 4 continued on next page. . .

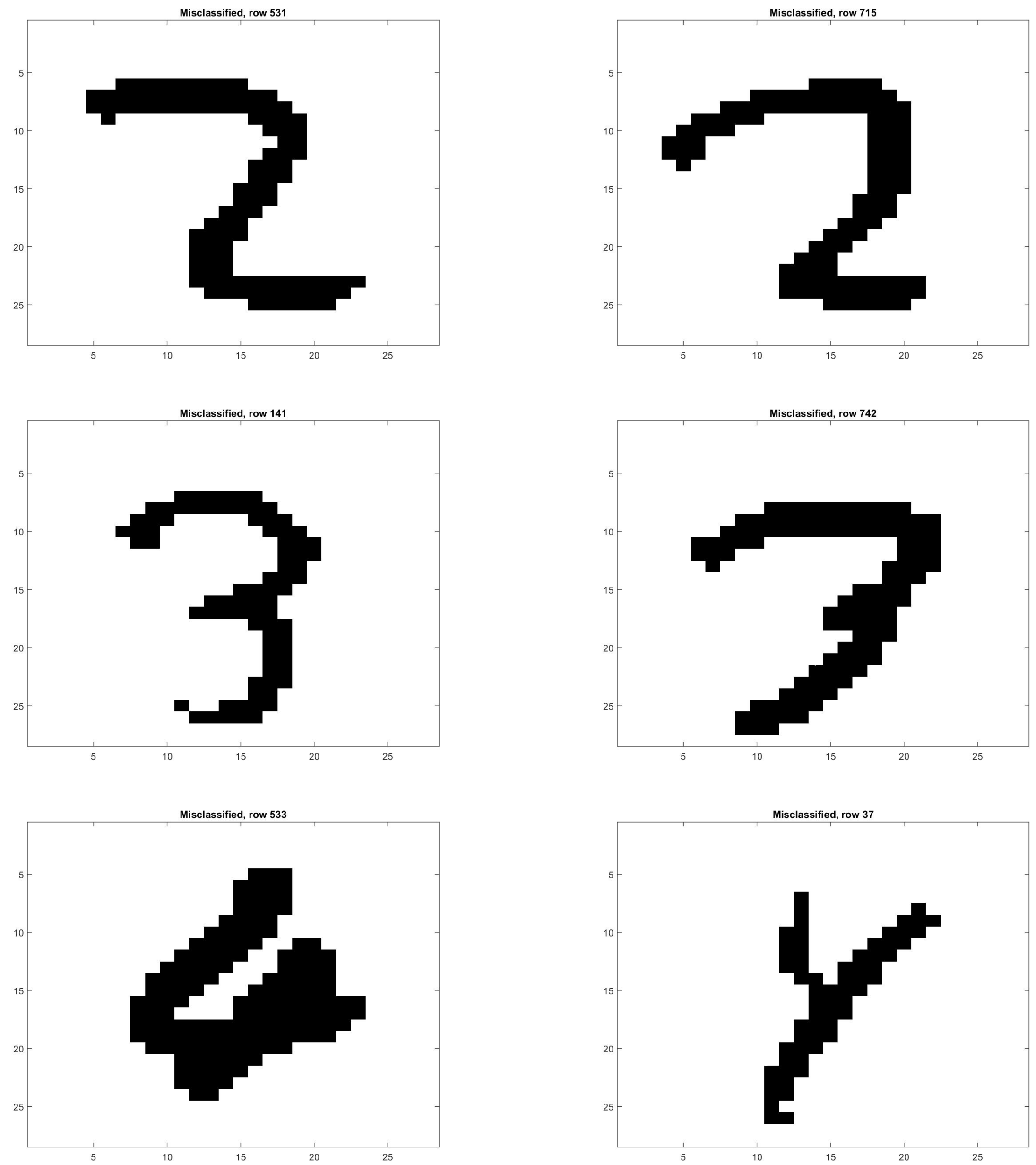


Figure 33: Some misclassified pictures

Problem 4 continued on next page. . .

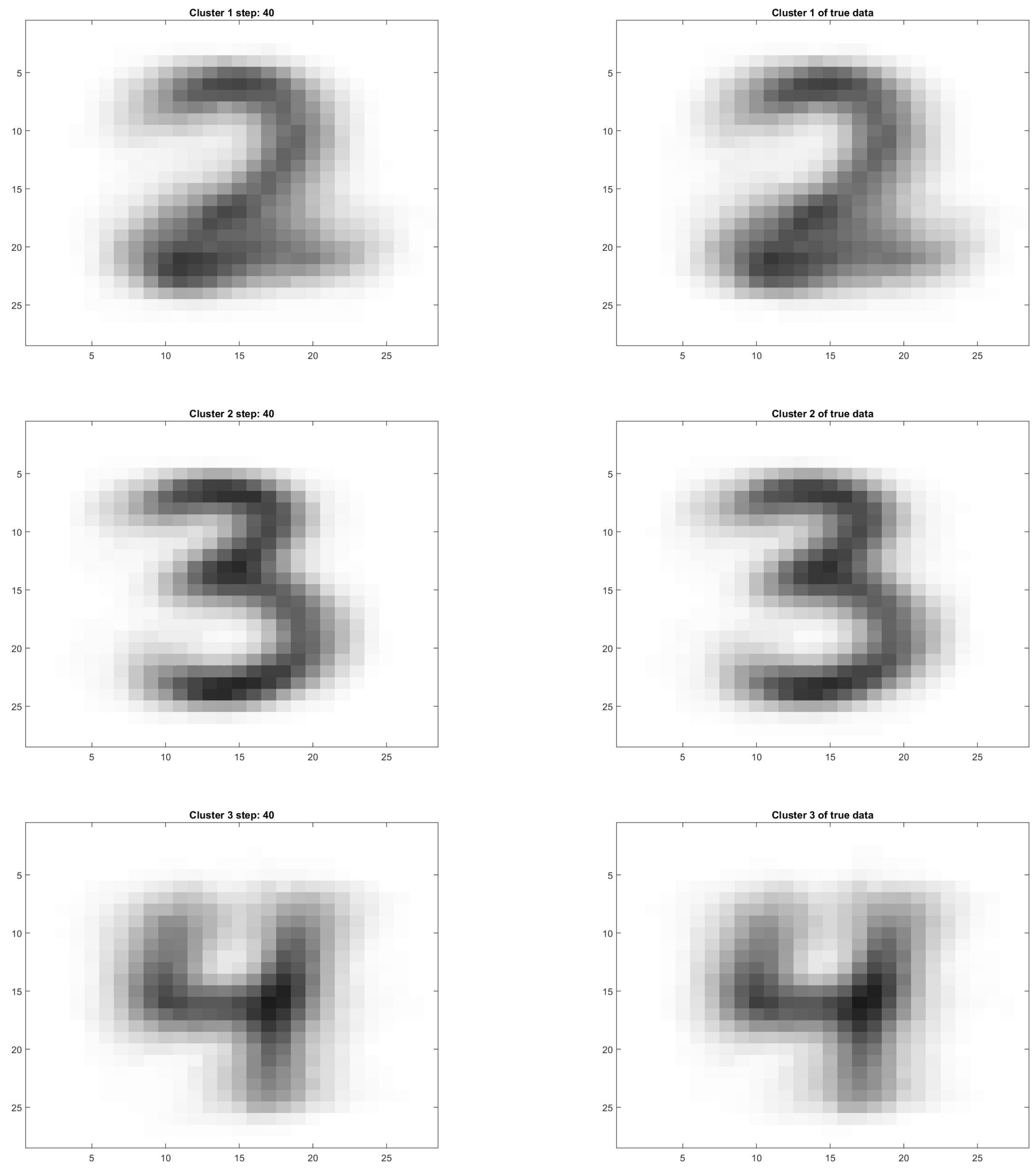


Figure 34: Clusters for $K = 3$, left: estimated with EM, right: true value

Problem 4 continued on next page. . .

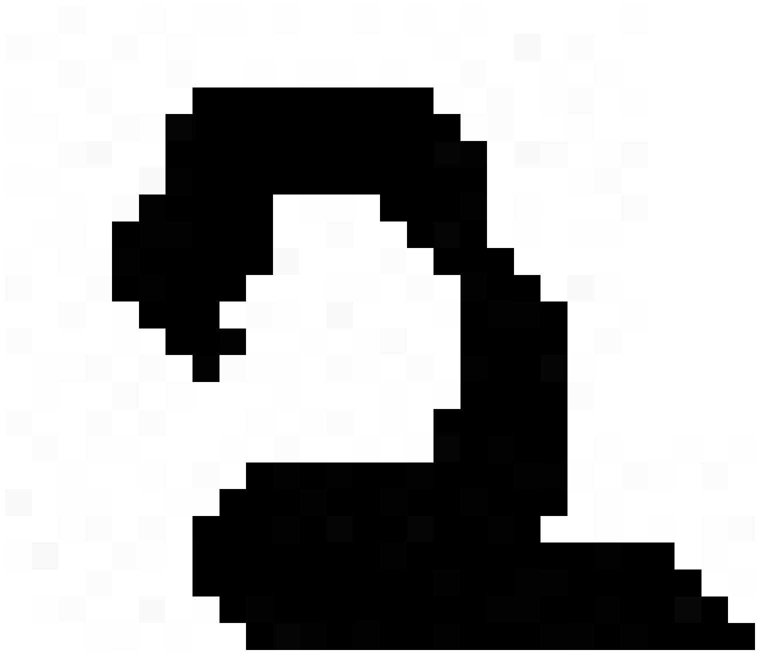


Figure 35: Handwritten digit of ′2”.

Problem 4 continued on next page. . .

APPENDIX

```
% %% Statistical Machine Learning Assignment 02
% % Simone Lederer    s1234567
% % Patrick Ebel      s4399803
%
% clear all;
% close all;
% clc;
%
% %% exercise I
%
% % please see the document
%
% %% exercise II
%
% % (1) get data points of an isotropic 2D-Gaussian
%
% [x,y] = meshgrid(-2:0.1:2,-2:0.1:2);
% X     = [x(:), y(:)];
% Y     = 3*mvnpdf(X, [0 0], 2/5 * eye(2));
% distr = reshape(Y, size(x));
%
% figure();
% surf(x,y,distr)
% xlabel('x_1'); ylabel('x_2'); zlabel('density');
% title('Surface plot of isotropic 2D-Gaussian')
%
%
% % (2) implement a 2-layer MLP
%
% % please see myMLP.m
%
% % initialize parameters with which to call the MLP
% hUnits = 8;          % specify number of hidden units
% eta = 0.1;           % learning rate
% cycles = 0;          % maximal number of cycles over training set
%
% % call the MLP
% y_test = myMLP(hUnits,X,Y,cycles,eta);
%
% figure();
% surf(x,y,reshape(y_test', size(x)))
% xlabel('x_1'); ylabel('x_2'); zlabel('density');
% title('Outputs of initialized/untrained MLP')


% (3)

% initialize parameters with which to call the MLP
hUnits = 8;          % specify number of hidden units
eta = 0.1;           % learning rate
cycles = 500;        % maximal number of cycles over training set

% call the MLP
y_test = myMLP(hUnits,X,Y,cycles,eta);

figure();
surf(x,y,reshape(y_test', size(x)))
xlabel('x_1'); ylabel('x_2'); zlabel('density');
title('Outputs of MLP trained for 500 cycles')


% (4)

% permute X and Y to a random order
permIdx = randperm(length(X));
X_perm = X(permIdx,:);
Y_perm = Y(permIdx);

% repeat training and show that convergence speed is faster:
% let myMLP run with same parameters as in 2.3 & compare development of error
y_test = myMLP(hUnits,X_perm,Y_perm,cycles,eta,permIdx);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% real data %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% (5)

% load and plot the real data set
data = load('a017_NNpdfGaussMix.txt', '-ASCII');
X = data(:,1:2); Y = data(:,3);

figure();
[x,y] = meshgrid(-2:0.1:2,-2:0.1:2);
surf(x,y,reshape(Y, size(x)))
xlabel('x_1'); ylabel('x_2'); zlabel('density');
title('Multimodal target probability density function')


% (6)

% initialize parameters with which to call the MLP
hUnits = 40;          % specify number of hidden units
eta = 0.01;           % learning rate
cycles = 2000;        % maximal number of cycles over training set

% permute X and Y to a random order
permIdx = randperm(length(X));
X = X(permIdx,:);
Y = Y(permIdx);

% train the MLP on the real data set
y_test = myMLP(hUnits,X,Y,cycles,eta,permIdx);

% un-permutate the outputs of the network to plot them in order
[~,unperm] = sort(permIdx);
y_test = y_test(unperm)';

figure();
[x,y] = meshgrid(-2:0.1:2,-2:0.1:2);
surf(x,y,reshape(y_test, size(x)))
xlabel('x_1'); ylabel('x_2'); zlabel('density');
title('Multimodal output probability density function')

% (7) BONUS

% initialize parameters with which to call the MLP
NIN = size(X,2);     % define dimension of input
hUnits = 40;         % specify number of hidden units
NOUT= size(Y,2);     % define dimension of output
FUNC='linear';       % define output function

% instantiate the network
NET = mlp(NIN, hUnits, NOUT, FUNC);

ALG = 'scg';         % define training mode: Conjugate Gradient Descent
```

```
OPTIONS=foptions;   % set options parameters to default values
OPTIONS(14) = 2000; % except for number of learning cycles, set to 2000

% learn the network weights
[NET, OPTIONS] = netopt(NET, OPTIONS, X, Y, ALG);

% forward propagate X through the learned NET to get predictions
predictions = mlpfwd(NET, X);

% assess goodness of predictions via SSE
SSE_netlab = 0.5 * sum((Y-predictions).^2);
```

```
function y_test = myMLP(M,X,Y,cycles,eta,permIdx)
% myMLP simulates a multilayer perceptron
%   M       - number of units in the hidden layer
%   X       - input values of data
%   Y       - target values of data
%   cycles  - number of times to loop through the training data
%   eta     - learning rate
%   mom     - momentum term (for gradient descent)

%% initialize the MLP

mom = 0;                         % set momentum to zero
D = size(X,2);                   % dimension of input
X = [X,ones(length(X),1)];       % concatenate 1's for bias input
D = D + 1;                       % incr. dimension of input (hidden layer bias)
M = M + 1;                       % incr. dimension of hidden layer (output bias)

% initialize weights in [-0.5,0.5] uniformly
w1=rand(D,M)-0.5;                        % weights of input -> layer1
w1(:,M) = 0;                             % (the weight to the bias is irrelevant)
w2=rand(M,1)-0.5;                        % weights of layer1 -> output

y_train = zeros(1,cycles*size(X,1));  % declare space to store development of outputs
E_train = zeros(1,cycles);            % declare space to store development of cost
dw_train = zeros(1,cycles);           % declare space to store development of change in
    weights

dw1 = zeros(M,D);   % initialize delta_weights layer 1
dw2 = zeros(1,M);   % initialize delta_weights output layer

% store outputs per cycles
y_cycle = zeros(1,size(X,1));
E_cycle = zeros(1,cycles);

for loops = 1:cycles       % loop through all cycles
    idx = 1;               % initialize/ reset number of iterations
    while (idx<=size(X,1))
        %% [forward propagate] activities, calculate error and cost

        %seq = randi(length(X));            % randomly choose input for stoch.grad.desc.
        seq = idx;

        a1 = X(seq,:)*w1; z  = tanh(a1);  % use tanh activation fct. for hidden units
        z(1,M) = 1;                         % set manually added hidden unit to 1 (bias)
        a2 = z*w2;          y  = a2;        % use linear activation fct. for output unit

        error = y - Y(seq,:);               % obtain errors and cost E (SSE)
        E     = 0.5 * sum(error.*error);    % at the output layer

        %% [back propagate] errors and calculate derivatives
        d2 = error;                         % back propagate to hidden layer, eq. (5.65)
        d1 = (1-z.^2)' .* (d2*w2);          % back propagate to input layer, eq. (5.66)

        dE2 = d2*z;                         % get error derivative, Bishop eq (5.53),
        dE1 = d1*X(seq,:);                  % analogous to Bishop eq (5.67)

        dw2 = eta * dE2 + mom * dw2;        % update weights, for mom = 0 this is
        dw1 = eta * dE1 + mom * dw1;        % equivalent to Bishop eq (5.41)

        %% calculate update in weights (set mom=0 for Bishop eq. (5.43))
        w2 = w2 - dw2';
        w1 = w1 - dw1';

        % update and store variables per learning step over all cycles
      y_train((loops-1)*size(X,1)+idx) = y;         % store regression outputs

        % update and store variables per learning step for each cycle
        dw  = max(max(max(abs(dw1))),max(abs(dw2))); % calculate maximum change of weights
        dw_train(loops)= dw_train(loops)+dw;          % store maximum adjustment of weights
        E_train(loops) = E_train(loops) + E;          % store cost (add up over all inputs)

        idx = idx+1;                                  % increment iteration counter
    end

        % feed forward the whole data set to get data for the plots
        for idx = 1:size(X,1)
            a1 = X(idx,:)*w1;
            z = tanh(a1); z(1,end) = 1;     % get hidden layer act.
            a2 = z*w2;                      % get output layer act.
            y_cycle(idx) = a2;
        end
        y_plotti = y_cycle;

        error = y_cycle' - Y;                        % obtain errors and cost E (SSE)
        E_cycle(loops)= 0.5 * sum(error.*error);     % at the output layer

        % every 20th cycle, do a surface plot
        if mod(loops,10) == 0
            % print the current error after each completed cycle
            fprintf('cycle %d; error %f; dw %f\n',loops,E_cycle(loops),dw_train(loops));

            if exist('permIdx','var')
                % un-permutate the outputs of the network to plot them in order
                [~,unperm] = sort(permIdx);
                y_plotti = y_cycle(unperm)';
            end

            %figure();
            [x,y] = meshgrid(-2:0.1:2,-2:0.1:2);
            surf(x,y,reshape(y_plotti, size(x)))
            xlabel('x_1'); ylabel('x_2'); zlabel('density');
            title(strcat('Outputs of MLP trained for', num2str(loops),'cycles'))
            drawnow
        end

end

%% do testing

y_test = nan(1,size(X,1));  % store regression outputs
E_test = nan(1,size(X,1));
jdx = 1;                    % initialize number of iterations
```

```

while (jdx<=length(X))
    %% [forward propagate] activities, calculate error and cost

    %seq = randi(length(X));             % randomly choose input for stoch.grad.desc.
    seq = jdx;

    a1 = X(seq,:)*w1; z  = tanh(a1);  % use tanh activation fct. for hidden units
    z(1,M) = 1;                        % set manually added unit to 1 (bias)
    a2 = z*w2;          y  = a2;        % use linear activation fct. for output unit

    error = y - Y(seq,:);              % obtain errors and cost E (SSE)
    E     = 0.5 * sum(error.*error);    % at the output layer

    % update and store variables
    y_test(jdx) = y;                                 % store regression outputs
    E_test(jdx) = E;                                 % store cost

    jdx = jdx+1;
end

%% return statistics and plot figures

if cycles > 0 % if training occured, plot training statistics
    figure()
    %loglog(E_cycle)
    plot(E_cycle)
    title(strcat('Training error as a function of cycles, final:',num2str(E_cycle(end))))
    xlabel('cycle')
    ylabel('training error (SSE)')

    figure()
    %loglog(dw_train)
    plot(dw_train)
    title(strcat('Change in weights as a function of cycles, final:',num2str(dw_train(end)))
    )
    xlabel('cycle')
    ylabel('max. weight change')
    fprintf('train error: %f \n',E_train(end));
    fprintf('dw: %f \n',dw_train(end));
end
end
```

```
%% Exercise 3: EM and Doping
%% Initialization
clear ; close all;

X = load('a011_mixdata.txt', '-ASCII');

[nrow, ncol] = size(X);
%% Ex3.1 EDA
% % eda(X);
% hist(X, 15);
% title('Historgram of all data')
% saveas(gcf, 'hist.jpeg')
% pause;
%
% scatter(X(:,1), X(:,2), 5, 'filled')
% hold on;
% scatter(X(:,3), X(:,4), 5, 'filled')
% legend('drug assumed to be present', 'drug assumend to be not present')
% title('Scatterplot of 2 groups')
% saveas(gcf, 'scatter_grouped.jpeg')
%% 3.2 EM algorithm - written in function EM.m

%% 3.3 K = 2, run EM
% K: number of classes assumed
K = 2;
fprintf(['The EM algorithm is running with K = ', num2str(K), '\n'])
% dimension of data used
D = ncol;
% initialize values pi, mu and Sigma
pi_0 = initPi(K);
[mu_0, Sigma_0] = initGaussian(X, K, D);

% evaluate logLikelihood of initial values
loglikeli_0 = logLikelihood(X, pi_0, mu_0, Sigma_0, K);

fprintf('Parameters are initialized.\n')
fprintf('Press any key to run the EM algorithm\n')
pause;

% minimal number of steps:
miniter = 100;

% Call function EM to run the EM algorithm
[loglikeli2, iter2, gamma2, pi2, mu2, Sigma2, convsteps2] = ...
    EM( X, K, pi_0, mu_0, Sigma_0, miniter, loglikeli_0);

%Visualize the data based on estimated class:
vizClass(X, gamma2, K);

fprintf(['The algorithm has converged after ', num2str(convsteps2),...
    ' steps.\n'])
fprintf('Press any key to continue with the correlation values\n');
pause;

% Get for each sample the most likely category:
[val, idx_2] = max(gamma2');
Z_2 = idx_2;

% Computation based on estimated Sigma (by EM())
rho12_comp1_K2 = rho12(Sigma2(1:D/2,1:2));
rho12_comp2_K2 = rho12(Sigma2(D+1:D+2,1:2));
isStrong_comp1_K2 = isStrong(rho12_comp1_K2);
isStrong_comp2_K2 = isStrong(rho12_comp2_K2);

test = [isStrong_comp1_K2, isStrong_comp2_K2] == 1;

fprintf(['Is correlation between x1 and x2 in the classes strong?: ',...
    num2str(test), '\n'])
fprintf('Press any key to run the EM algo again on K=3\n')
pause;
%% 3.4
K = 3;
fprintf(['The EM algorithm is running with K = ', num2str(K), '\n'])

pi_0 = initPi(K);
[mu_0, Sigma_0] = initGaussian(X, K, D);

% evaluate logLikelihood of initial values
loglikeli_0 = logLikelihood(X, pi_0, mu_0, Sigma_0, K);

% minimal number of steps:
```

```
miniter = 100;

% Call function EM to run the EM algorithm
[loglikeli3 , iter3 , gamma3, pi3 , mu3, Sigma3, convsteps3] = ...
    EM( X, K, pi_0 , mu_0, Sigma_0 , miniter , loglikeli_0);

%Visualize the data based on estimated class:
vizClass(X, gamma3, K);

fprintf(['The algorithm has converged after ', num2str(convsteps3) ,...
     ' steps.\n'])
fprintf('Press any key to continue with the correlation values\n');
pause;

% Correlation coefficients
[val , idx_3] = max(gamma3');
Z_3 = idx_3;

rho12_comp1_K3 = rho12(Sigma3(1:2 ,1:2));
rho12_comp2_K3 = rho12(Sigma3(ncol+1:ncol+2,1:2));
rho12_comp3_K3 = rho12(Sigma3(2*ncol+1:2*ncol+2,1:2));
isStrong_comp1_K3 = isStrong(rho12_comp1_K3);
isStrong_comp2_K3 = isStrong(rho12_comp2_K3);
isStrong_comp3_K3 = isStrong(rho12_comp3_K3);

test = [isStrong_comp1_K3 , isStrong_comp2_K3 , isStrong_comp3_K3] == 1;

fprintf(['Is correlation between x1 and x2 in the classes strong?: ',...
    num2str(test), '\n'])
fprintf('Press any key to run the EM algo again on K=4\n')
pause;

K = 4;
fprintf(['The EM algorithm is running with K = ', num2str(K), '\n'])

pi_0 = initPi(K);
[mu_0, Sigma_0] = initGaussian(X, K, D);

% evaluate logLikelihood of initial values
loglikeli_0 = logLikelihood(X, pi_0 , mu_0, Sigma_0 , K);

% minimal number of steps:
miniter = 100;

% Call function EM to run the EM algorithm
[loglikeli4 , iter4 , gamma4, pi4 , mu4, Sigma4, convsteps4 ] = ...
    EM( X, K, pi_0 , mu_0, Sigma_0 , miniter , loglikeli_0);

%Visualize the data based on estimated class:
vizClass(X, gamma4, K);

fprintf(['The algorithm has converged after ', num2str(convsteps4) ,...
     ' steps.\n'])
fprintf('Press any key to continue with the correlation values\n');
pause;

% Correlation coefficients
[val , idx_4] = max(gamma4');
Z_4 = idx_4;

% comp1 = Z_4==1;
% comp2 = Z_4==2;
% comp3 = Z_4==3;
% comp4 = Z_4==4;
% % Correlation coefficient
% rho_12_comp1_K4 = rho12(X(comp1,1), X(comp1,2));
% rho_12_comp2_K4 = rho12(X(comp2,1), X(comp2,2));
% rho_12_comp3_K4 = rho12(X(comp3,1), X(comp3,2));
% rho_12_comp4_K4 = rho12(X(comp4,1), X(comp4,2));

rho12_comp1_K4 = rho12(Sigma4(1:2,1:2));
rho12_comp2_K4 = rho12(Sigma4(ncol+1:ncol+2,1:2));
rho12_comp3_K4 = rho12(Sigma4(2*ncol+1:2*ncol+2,1:2));
rho12_comp4_K4 = rho12(Sigma4(3*ncol+1:3*ncol+2,1:2));
isStrong_comp1_K4 = isStrong(rho12_comp1_K4);
isStrong_comp2_K4 = isStrong(rho12_comp2_K4);
isStrong_comp3_K4 = isStrong(rho12_comp3_K4);
isStrong_comp4_K4 = isStrong(rho12_comp4_K4);

test = [isStrong_comp1_K4, isStrong_comp2_K4, isStrong_comp3_K4, ...
    isStrong_comp4_K4] == 1;

fprintf(['Is correlation between x1 and x2 in the classes strong?: ',...
    num2str(test), '\n'])

fprintf(['The 1st group (with high correlation of x1 and x2)'...
    'has weight ', num2str(pi4(test)), '\n'])
pause;
%% 3.5 Identify drug X and tamper
A = [11.85,2.2,0.5,4.0];
B = [11.95,3.1,0.0,1.0];
C = [12.00,2.5,0.0,2.0];
D = [12.00,3.0,1.0,6.3];

K=4;
[gamma_new, z_new] = catNewData([A;B;C;D], pi4, mu4, Sigma4, K);

fprintf(['A, B, C and D belong to groups ',num2str(z_new),...
    ' respectively.\n'])
```

```
function [ gamma, z ] = catNewData( data, pi, mu, Sigma, K )
%catNewData calculates the probabilities of belonging to the classes
% of new data
%   The posiblity to belong to a class and the final categorization based
%   on the maximal probability are handed back. The probabilities are
%   calcualted with the estiamted densitities (output by EM)

 [nrow, ncol] = size(data);
 D = ncol;
% E-step:
 gamma = zeros(nrow,K);
 prob = zeros(K,1);
 for n =1:nrow
     for k = 1:K
         prob(k) = mvnpdf(data(n,:), mu(k,:), Sigma(((k-1)*D+1:k*D),:));
     end
     zaehler = pi.*prob;
     nenner = pi'*prob;
    gamma(n,:) = zaehler./nenner;
 end

 [val idx] = max(gamma');
 z = idx;

```

```
end
```

```
function [ X ] = eda( X )
%eda(X) performs a visualization analysis of the data set X
%   histograms and 2D and 3D scatterplots are created and saved in the
%   folder figures
mkdir('figures')

hist(X, 15)
title('Historgram of all data')
saveas(gcf, 'figures/hist.jpeg')
pause;
for i = 1:4
    hist(X(:,i), 15)
    xlabel(['x', num2str(i)]);
    title(['Historgram of x', num2str(i)])
    saveas(gcf, ['figures/hist_', num2str(i), '.jpeg'] )
    pause;
    for j = i:4
        if i ~= j
        scatter(X(:,i), X(:,j), 5, 'filled');
        xlabel(['x', num2str(i)]); ylabel(['x', num2str(j)]);
        title(['Scatterplot of x', num2str(i), ' and x', num2str(j)])
        saveas(gcf, ['figures/scatter_', num2str(i), '_', num2str(j), '.jpeg'] )
        pause;
        end
    end
end

scatter3(X(:,1), X(:,2), X(:,3), 5, 'filled')
xlabel('x1'); ylabel('x2'); zlabel('x3');
title('Scatterplot of x1, x2, x3')
saveas(gcf, 'figures/scatter3_x1_x2_x3.jpeg')
pause;
scatter3(X(:,1), X(:,2), X(:,4), 5, 'filled')
xlabel('x1'); ylabel('x2'); zlabel('x4');
title('Scatterplot of x1, x2, x4')
saveas(gcf, 'figures/scatter3_x1_x2_x4.jpeg')
pause;
scatter3(X(:,1), X(:,3), X(:,4), 5, 'filled')
title('Scatterplot of x1, x3, x4')
xlabel('x1'); ylabel('x3'); zlabel('x4');
saveas(gcf, 'figures/scatter3_x1_x3_x4.jpeg')
pause;
scatter3(X(:,2), X(:,3), X(:,4), 5, 'filled')
xlabel('x2'); ylabel('x3'); zlabel('x4');
title('Scatterplot of x2, x3, x4')
saveas(gcf, 'figures/scatter3_x2_x3_x4.jpeg')
end
```

```
function [ loglikeli, iter, gamma, pi, mu, Sigma, convsteps ] = EM( X, K, pi_0, mu_0,
    Sigma_0, ...
    miniter, loglikeli_0)
%EM algorithm applied to data X with K-Gaussian mixture model
%   Detailed explanation goes here

if miniter < 100
    error('miniter: minimal number of steps must be > 100');
else
    [nrow, D] = size(X);
    iter = 1;
    mu = mu_0;
    Sigma = Sigma_0;
    pi = pi_0;
    % Z = zeros(nrow, K);
    loglikeli = loglikeli_0;
    eps = 1;
    while( iter < miniter || abs(loglikeli(iter)-loglikeli(iter-1))> eps)
        %E-step
        gamma = zeros(nrow,K);
        prob = zeros(K,1);
        for n =1:nrow
            for k = 1:K
                prob(k) = mvnpdf(X(n,:), mu(k,:), Sigma(((k-1)*D+1:k*D),:));
            end
            zaehler = pi.*prob;
            nenner = pi'*prob;
            gamma(n,:) = zaehler./nenner;
        end

        %M-step
        N = sum(gamma);
    %     returns a K x D vector with each row containing mu_k
        mu_new = 1./(ones(D,1)*N)' .* (gamma'*X);
    %     returns a K*D x D matrix with each ((k-1)*D+1)st until k*Dth row
    %     containing Sigma_k
        Sigma_new = zeros(K*D, D);
        for k = 1:K
             Sigma_new(((k-1)*D+1:k*D),:) = 1/N(k) * ...
                (gamma(:,k) * ones(1,D).* ...
                (X - ones(nrow,1)*mu_new(k,:)))'* ...
                (X - ones(nrow,1)*mu_new(k,:));
        end
        pi_new = N./nrow;

    %     Evaluate log_likelihood
        loglikeli = [loglikeli; ...
            logLikelihood(X, pi_new, mu_new, Sigma_new, K)];
        if (iter>1 && abs(loglikeli(iter)-loglikeli(iter-1)) < eps)
            convsteps = iter;
        end
        iter = iter+1;
        loglog(loglikeli, '-b');
        title(['Log-log-development of loglikelihood with K=', num2str(K)]);
        xlabel('log(iter)'); ylabel('log(loglikeli)');
        drawnow;
        pi = pi_new';
        mu = mu_new;
        Sigma = Sigma_new;
    end
    saveas(gcf, ['loglikelihood_K', num2str(K), '.jpeg'])
end
end
```

```
function [ mu_0, Sigma_0 ] = initGaussian( X, K, D )
%initGaussian calcualtes initial mu_0, and Sigma_0 to start with the EM
rng('default');
rng(5);
mu_0 = zeros(K,D);
sigma_0 = zeros(K,D);
Sigma_0 = zeros(K*D, D);
for k=1:K
    eps = -1 + 2*rand();
    mu_0(k,:) = mean(X) + eps;
    sigma_0(k,:) = 4*rand()+2;
    Sigma_0(((k-1)*D+1:k*D),:) = diag(sigma_0(k,:));
end

end
```

```
function [pi] = initPi( K )
% initPi calculates initiative values of weights necessary
% to start with the EM

% K: number of classes

pi = zeros(K,1);
for k=1:K
    pi(k) = 1/K;
end

end
```

```
function [ t ] = isStrong( rho12 )
% Test whether the correlation between X1 and X2 is strong

% get absolute correlation value between x1, x2:
val = rho12;
if val > 0.5
    t = 1;
else
    t = 0;
end

end
```

```
function [ likeli ] = logLikelihood( X, pi, mu, Sigma, K)
%logLikelihood
%   for a D-dim data set X the likelihood of its Gaussian mixture
%   with K classes is given with weights pi and the Gaussian
%   distributions defined mu and sigma
    [N, D] = size(X);
    sum = 0;
    for i=1:N
        term = 0;
        for k=1:K
            prob = mvnpdf(X(i,:), mu(k,:), Sigma((k-1)*D+1:k*D,:));
            term = term + pi(k)*prob;
        end
        sum = sum + log(term);
    end
    likeli = sum;
end
```

```
function [ rho12 ] = rho12(Sigma)
%rho12 calculates the correlation between 2 vectors X and Y
%     rho12 = cov(X, Y)./ sqrt(var(X)*var(Y));

%     test
    co = Sigma;
    rho12 = co(1,2)/sqrt(co(1,1)*co(2,2));
end
```

```
function [  ] = vizClass( X, gamma, K )
%vizClass
%Visualize the data based on estimated classes:
```

```
nrow = length(X);
[val idx] = max(gamma');
Z = idx;
colormap('jet')
% cols = ['r', 'b', 'k', 'c'];
% cols2 = ones(nrow,1)*cols(1:K);
% col3 = diag(cols2(:,Z));
scatter(X(:,1),X(:,2), 10, Z, 'filled');
xlabel('x1'); ylabel('x2');
title('Scatterplot of x1 and x2 coloured by class');
drawnow;
filename = ['classified_x1_x2_K', num2str(K), '.jpeg'];
saveas(gcf, filename);

end
```

```
%% Exercise 4: Handwritten Digit Recognition
%% Initialization
clear ; close all;

N = 800; D = 28*28; X = zeros(N,D);
fid = fopen ('a012_images.dat', 'r');
for i = 1:N
X(i,:) = fread(fid, [D], 'uint8');
end;
status = fclose(fid);
X = double(X);
%% Ex4.1 EDA
% image()
rng('default');
numb_eda = randi(801, 6, 1);
BW_map=[1, 1, 1; 0, 0, 0];

for i=1:length(numb_eda)
    bild1 = zeros(28);
    for j=1:28
        bild1(j,:) = X(i,(j-1)*28+1:(j-1)*28+28)*100;
    end
    figure;
    image(bild1');
    colormap(BW_map);
    title(['row ', num2str(numb_eda(i))]);
    saveas(gcf, ['image_', num2str(i),'.jpeg'])
end

fprintf(['The data is loaded and five randomly chosen plots ',...
    'of the data are displeyed in figures\n']);
fprintf('Hit any key to run the EM on X with 3 clusters\n');
pause;
close all;
%% Ex4.2 EM for Bernoulli - written in EM_bernoulli()X,K,pi,mu,miniter)...
%                                 and initPi_Ex4(K).

%% Ex4.3 Run EM on X
K=3;

pi_0 = initPi_Ex4(K);
mu_0 = 0.25 + rand(K,28*28)*0.5; %K x 28*28 vector
% mu_0 = mu_0./(sum(mu_0,2)*ones(1,28*28));

miniter = 40;
[gamma3, pi3, mu3] = EM_bernoulli(X,K,pi_0,mu_0,miniter,'estim');
```

```

fprintf('The EM on X has run with 3 clusters assumed.\n')
fprintf('Hit any key to run the EM on X with 2 clusters.\n');
pause;
%% Ex4.4 run EM with different K, different initiatizations and compare

K=2;
pi_0 = initPi_Ex4(K);
mu_0 = 0.25 + rand(K,28*28)*0.5; %K x 28*28 vector
miniter = 40;
[~, ~, ~] = EM_bernoulli(X,K,pi_0,mu_0,miniter,'estim');

fprintf('The EM on X has run with 2 clusters assumed.\n');
fprintf('Hit any key to run the EM on X with 4 clusters\n');
pause;

K=4;
pi_0 = initPi_Ex4(K);
mu_0 = 0.25 + rand(K,28*28)*0.5; %K x 28*28 vector
miniter = 40;
[gamma4, pi4, mu4] = EM_bernoulli(X,K,pi_0,mu_0,miniter,'estim');

fprintf('The EM on X has run with 4 clusters assumed.\n');
fprintf('Hit any key to identify misclassified images\n');

% Compare erstimated groups with real labels
% open labels
fid = fopen('a012_labels.dat', 'r');
Z = fread(fid, N, 'uint8');

% get grouping from estimation with 3 clusters
[~, idx_3] = max(gamma3);
Z_3 = idx_3';
% get classified digits and label them
digit2 = (Z_3 == 1)*2;
digit3 = (Z_3 == 3)*3;
digit4 = (Z_3 == 2)*4;
% vector with labels of classified data:
Z_3_new = digit2 + digit3 + digit4;
% mis ~= 0: misclassified data
mis = (Z - Z_3_new);
% which entries are misclassified
d2_mis = (mis.*(Z==2));
d3_mis = (mis.*(Z==3));
d4_mis = (mis.*(Z==4));
% get indices of misclassified images
d2_misind = find(d2_mis~=0);
d3_misind = find(d3_mis~=0);
d4_misind = find(d4_mis~=0);
% get number of misclassified images per cluster
missed = [sum(d2_mis ~= 0), sum(d3_mis ~= 0), sum(d4_mis ~= 0)];
fprintf(['# of misclassified pictures 2, 3 and 4 are ',...
    num2str(missed), '\n'])
fprintf(['while the true weights are ', num2str(pi3'), '\n'])
fprintf('Hit any key to print some the misclassified pictures\n');
pause;
% draw 2 miclassified pictures of each class randomly
rng('default');
numb2 = randi(sum(d2_mis ~= 0), 2, 1);
numb3 = randi(sum(d3_mis ~= 0), 2, 1);
numb4 = randi(sum(d4_mis ~= 0), 2, 1);

```

```
numb = [d2_misind(numb2); d3_misind(numb3); d4_misind(numb4)];
BW_map=[1, 1, 1; 0, 0, 0];
% Z(numb);Z_3_new(numb);
% create images of misclassified digits
for i=1:length(numb)
    bild1 = zeros(28);
    for j=1:28
        bild1(j,:) = X(numb(i),(j-1)*28+1:(j-1)*28+28)*100;
    end
    figure;
    image(bild1');
    colormap(BW_map);
    title(['Misclassified, row ', num2str(numb(i))]);
    saveas(gcf, ['Misclassified_image_', num2str(i),'.jpeg'])
end

fprintf('Hit any key to initialize the classes with true values\n');
pause;

K=3;
nrow = length(Z);
pi_true = [sum(Z==2), sum(Z==3), sum(Z==4)]./nrow;
% there are no 'true' values for mu - start with empirical mean
mu_true = zeros(K, 28*28);
for k=1:K
    mu_true(k,:) = mean(X(Z==(k+1),:));
end

close all;
bild1 = zeros(28);
mu_mapped = round((1 - mu_true)*255);
for i=1:K
    for j=1:28
        bild1(j,:) = mu_mapped(i,(j-1)*28+1:(j-1)*28+28);
    end
    figure;
    image(bild1');
    colormap(gray(255));
    title(['Cluster ', num2str(i), ' of true data']);
    drawnow;
    saveas(gcf, ['Val=true_cluster_', num2str(i), '_K=',...
        num2str(K), '.jpeg']);
end
[gamma_true_estim, pi_true_estim, mu_true_estim] = ...
    EM_bernoulli(X, K, pi_true, mu_true, miniter, 'true');

[~, idx_3] = max(gamma_true_estim);
Z_true = idx_3+1;

mis_true = sum(Z_true' ~= Z);
fprintf('The EM on X has run with true values.\n')
fprintf([num2str(mis_true), ' pictures are still misclassified', '\n']);
fprintf('Hit any key to run the EM on an own handwritten letter\n');
pause;
%% Ex4.5 - create own handwritten digit image
own_pic = imread('handwritten-digit/2_v04.bmp'); %reads a 28*28 picture
own_pic = 1 - own_pic';
own_pic_vec = zeros(1, 28*28);
for i=1:28
    own_pic_vec((1 + (i-1)*28):i*28) = own_pic(i,:);
end
% Visualize digit
```

```
BW_map=[1, 1, 1; 0, 0, 0];
figure;
bild1 = zeros(28);
for j=1:28
    bild1(j,:) = own_pic_vec((j-1)*28+1:(j-1)*28+28)*100;
end
image(bild1');
colormap(BW_map);


[gamma_own_pic, z_own_pic] = catNewData_Ex4( own_pic_vec, pi3, mu3, 3 );

fprintf(['Handwritten digit 2 is classified into cluster ',...
    num2str(z_own_pic)])
```

```
function [ gamma, z ] = catNewData_Ex4( data, pi, mu, K )
%catNewData calculates the probabilities of belonging to the classes
% of new data
%   The posiblity to belong to a class and the final categorization based
%   on the maximal probability are handed back. The probabilities are
%   calcualted with the estiamted densitities (output by EM)

[nrow, ncol] = size(data);
D = ncol;
% E-step:

% prob = zeros(K,1);
%
% for k = 1:K
%     prob(k,:) = prod(((ones(nrow,1)*mu(k,:)).^double(X) .* ...
%     (1- (ones(nrow,1)*mu(k,:))).^double(X)), 2) ;
% end
% nenner = (prob'*pi); %(nrow x 1) matrix
% zaehler = ones(nrow,1)*pi'.*prob'; % (nrow x K)- matrix
% gamma = (zaehler) ./ (nenner*ones(1,K));
prob = zeros(K,1);
prob_log = zeros(K,1);
for k = 1:K
%   prob_log_k = zeros(nrow,1);
   prob_k = zeros(1, D);
   for d = 1:D
       prob_k(d) = data(d)*log(mu(k,d) + 0.001) ...
           + (1 - data(d))*log(1 - mu(k,d));
   end
   prob_log(k,:) = sum(prob_k(:)) + log(pi(k));
   prob(k) = exp(prob_log(k,:));
end


denom = sum(prob);
gamma = prob ./ (ones(K,1)*denom);




[val, idx] = max(gamma');
z = idx;

end
```

```
function [ gamma, pi, mu] = EM_bernoulli( X, K,...
    pi_0, mu_0, miniter, name)
%EM algorithm applied to data X with K-Bernoullin mixture model
```

```matlab

if miniter<40
    error('miniter: minimal number of steps must be > 40');
else
    [nrow, D] = size(X);
    iter = 1;
    mu = mu_0; % K x 28*28 vector
    pi = pi_0;
    while( iter < miniter)
        %E-step
        prob = zeros(K,nrow);
        prob_log = zeros(K,nrow);
        for k = 1:K
            prob_log_k = zeros(nrow,1);
            prob_k_n = zeros(nrow, D);
            for i = 1:nrow
                for d = 1:D
                    prob_k_n(i,d) = X(i,d)*log(mu(k,d) + 0.001) ...
                        + (1 - X(i,d))*log(1 - mu(k,d));
                end
                prob_log_k(i) = sum(prob_k_n(i,:)) + log(pi(k));
            end
            prob_log(k,:) = prob_log_k;
            prob(k,:) = exp(prob_log(k,:));
        end
%         prob = exp(prob_log);

        denom = sum(prob);
        gamma = prob ./ (ones(K,1)*denom);
        %M-step
        N_K = sum(gamma,2);
        mu_new = 1./N_K*ones(1,28*28) .* (gamma*X);
%         pi_new = N./nrow;
%         pi_new_log = N_log./nrow;
        pi_new = N_K./nrow;

        iter = iter+1;

        pi = pi_new;
        mu = mu_new;

        % Visualize mu:
        close all;
        bild1 = zeros(28);
        mu_mapped = round((1 - mu)*255);
        for i=1:K
            for j=1:28
                bild1(j,:) = mu_mapped(i,(j-1)*28+1:(j-1)*28+28);
            end
            figure;
            image(bild1');
            colormap(gray(255));
            title(['Cluster ', num2str(i), ' step: ', num2str(iter)]);
            drawnow;
        end
    end
    % Visualize mu for savinf:
    close all;
    bild1 = zeros(28);
    mu_mapped = round((1 - mu)*255);
    for i=1:K
        for j=1:28
```

```
            bild1(j,:) = mu_mapped(i,(j-1)*28+1:(j-1)*28+28);
        end
        figure;
        image(bild1');
        colormap(gray(255));
        title(['Cluster ', num2str(i), ' step: ', num2str(iter)]);
        drawnow;
        saveas(gcf, [name, '_cluster_', num2str(i), '_K=', num2str(K), '.jpeg']);
    end
end
end
```

```
function [pi] = initPi_Ex4( K )
% initPi calculates initiative values of weights necessary
% to start with the EM

% K: number of classes

pi = zeros(K,1);
% pi(1) = 0.2;
for k=1:K
    pi(k) = 1/K;
end

end
```