Assignment 4

Juraj Šušnjara (s4846559)
Sebastian Tiesmeyer (s4373162)

November 2016

1 Exercise 1 - Gaussian processes for regression

We consider a noisy model of the form:

$$t_n = y_n + \epsilon_n \tag{1}$$

Noise process has a Gaussian distribution:

$$p(t_n|y_n) = \mathcal{N}(t_n|y_n, \beta^{-1}) \tag{2}$$

We will use following kernel function for Gaussian process regression:

$$k(\boldsymbol{x}, \boldsymbol{x'}) = \theta_0 exp(-\frac{\theta_1}{2}||\boldsymbol{x} - \boldsymbol{x}'||^2) + \theta_2 + \theta_3 \boldsymbol{x}^T \boldsymbol{x}' \tag{3}$$

1.1

Implementation of equation 3 in MATLAB is:

```
function k = kernel( x, xt, theta )

    k = theta(1).*exp((-theta(2)./2).*norm(x-xt).^2)+
                     theta(3)+theta(4).*(x'*xt);

end
```

1.2

Following script is used for computation of Gram matrix $\boldsymbol{K}(\boldsymbol{X}, \boldsymbol{X})$:

```
N = 101;
X = linspace(-1,1,N);
K = zeros(N, N);
theta = [1 1 1 1];

for i=1:N
    for j=1:N
        K(i,j) = kernel(X(i), X(j), theta);
    end
end
```

1.3

The size of $\boldsymbol{K}$ is $101 * 101$, $d = R^2$.

A given [m*m] matrix M is semipositive definite if:

$$z^T M z \geq 0 \tag{4}$$

for every given [m*1] vector z.

The Gram matrix is the cross product of all vectors of the vector set X:

$$G = X^T X \tag{5}$$

Substituting G in equation (4) results in a *squared* absolute of the cross product of X and z, which is always zero or positive:

$$z^T X^T X z = ||Xz||_2^2 \geq 0 \tag{6}$$

1.4

Plots of 5 random samples produced from Gaussian process prior $y(\boldsymbol{X}) \propto \mathcal{N}(0, \boldsymbol{K(X, X)})$ are presented on figure 1.

1.5

On figure 2 we can see different plots produced from a Gaussian process prior with different $\boldsymbol{\theta}$ values. Plots are drawn by following MATLAB code snippet:

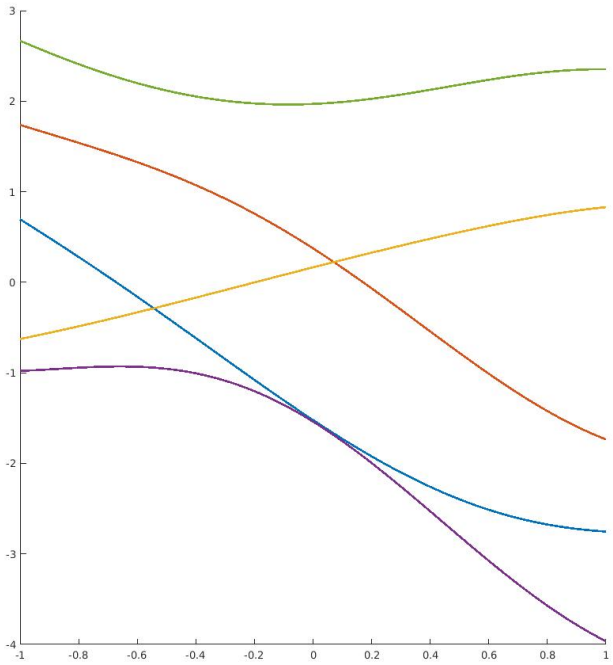
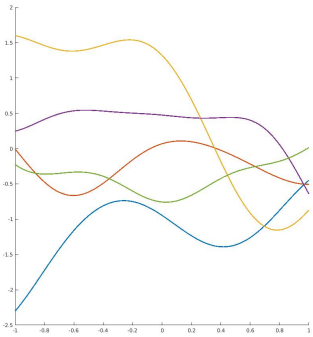


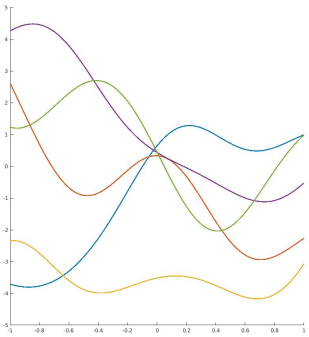
Figure 1: Plots of 5 different samples from Gaussian process prior where $\boldsymbol{\theta} = (1, 1, 1, 1)$

```
mu = zeros(101,1);
Y = mvnrnd(mu, K);
plot(X,Y);
```

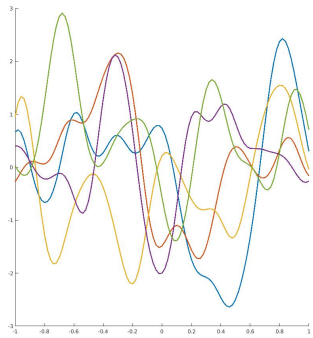
Bonus: The different θs emphasize or suppress different parts of the kernel equation which leads to different covariance matrices. θ_2 is just a constant that raises the value of every element, so the covariance of all (x_1, x_2)-combinations increases regardless of their euclidean distance. The sample curves will show small local and global covariances, be relatively leveled, flat and smooth. θ_0 and θ_1 both influence the first term. θ_0 increases covariance proportionally, so local and global variances in/decrease the same and the range of y(x) is basically scaled up. θ_1 decreases the local variance by magnifying the absolute distance indicator $||x - x'||$ ($- >$when [-0.1:0.1] were scaled up in Bishop 6.5 [3], we would retrieve the original function). θ_3 promotes global devariance of (x_1, x_2)-combinations with opposite signs.



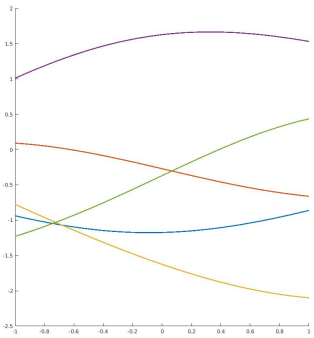
(a) $\boldsymbol{\theta} = (1, 4, 0, 0)$



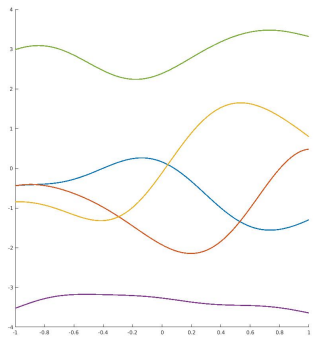
(b) $\boldsymbol{\theta} = (9, 4, 0, 0)$



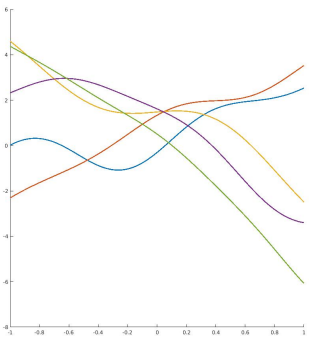
(c) $\boldsymbol{\theta} = (1, 64, 0, 0)$



(d) $\boldsymbol{\theta} = (1, 0.25, 0, 0)$



(e) $\boldsymbol{\theta} = (1, 4, 10, 0)$



(f) $\boldsymbol{\theta} = (1, 4, 0, 5)$

Figure 2: Plots of samples from Gaussian process prior with different $\boldsymbol{\theta}$ values

1.6

Covariance matrix $\boldsymbol{C}$ corresponding to the marginal distribution of the training target values $p(\boldsymbol{t}) = \mathcal{N}(\boldsymbol{t}|\boldsymbol{0}, \boldsymbol{C})$ is calculated using the expression 7 and shown in epression 8

$$C(x_n, x_m) = k(x_n, x_m) + \beta^{-1}\delta_{nm} \tag{7}$$

$$C = \begin{pmatrix} 3.25 & 1.68 & 1.58 & 1.97 \\ 1.68 & 3.04 & 2.06 & 1.94 \\ 1.58 & 2.06 & 3.09 & 1.89 \\ 1.97 & 1.94 & 1.89 & 3.01 \end{pmatrix} \tag{8}$$

1.7

To find the conditional distribution $p(t_{N+1}|\boldsymbol{t})$ we start with writing joint distribution $p(\boldsymbol{t}_{N+1})$ which is shown in expression 9.

$$p(\boldsymbol{t}_{N+1}) = \mathcal{N}(\boldsymbol{t}_{N+1}|\boldsymbol{0}, \boldsymbol{C}_{N+1}) \tag{9}$$

where

$$\begin{aligned} \boldsymbol{C}_{N+1} &= \begin{pmatrix} \boldsymbol{C}_N & \boldsymbol{k} \\ \boldsymbol{k}^T & c \end{pmatrix} \\ c &= k(x_{N+1}, x_{N+1} + \beta^{-1}) \\ \boldsymbol{k} &= k(x_n, x_{N+1}) \end{aligned} \tag{10}$$

Finally, mean and covariance are calculated by following expressions:

$$m(x_{N+1}) = \boldsymbol{k}^T \boldsymbol{C}_N^{-1} \boldsymbol{t} \tag{11}$$

$$\sigma^2(x_{N+1}) = c - \boldsymbol{k}^T \boldsymbol{C}_N^{-1} \boldsymbol{k} \tag{12}$$

Based on given data set, mean and covariance of a new target value t corresponding to input $x = 0$ are given in expressions 13 and 14.

$$m(0) = -0.0252 \tag{13}$$

$$\sigma(0) = 1.2357 \tag{14}$$

1.8

This depends on the mean of the input data - t needs to have mean zero as well. To achieve this (and we want to, since our assumption a priori is that the target process = mean(0)), one can adjust $f(x)$ by setting θ_2 to be -mean(t).

2 Exercise 2 - Neural network regression

2.1 Plotting a 3D Gaussian

MATLAB code to plot a 3D Gaussian with $\mu = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$ and $\Sigma = \begin{pmatrix} 0.4 & 0 \\ 0 & 0.4 \end{pmatrix}$

```
figure(1)
clf;

%Initiate X Y values and covmat:
[X,Y] = meshgrid(-2:.1:2);
Sigma = (2/5)*eye(2);

%function handle to calculate a target value:
tgt = @(X,Y) reshape(mvnpdf([X(:) Y(:)],[0 0],Sigma),size(X));

%Calculate Z-values & plotification:
Z= reshape(tgt(X(:),Y(:)),size(X))
surf(X,Y,Z)
```

Output:

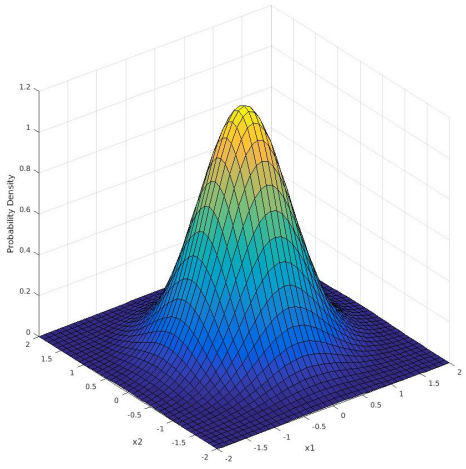


Figure 3: Plot of an isotropic 2D Gaussian $y = 3 \cdot \mathcal{N}(\boldsymbol{x}|0, \frac{2}{5}I_2)$

2.2 Initiating a network

```
%%%  --- Implement a 2-layer ANN: ---
M=8;         %hidden layer size
eta = .1;    %learning rate

%function handles for forward propagation:
fwdp1 =@(x,w)  tanh(([x,1])*w);
fwdp2 =@(a,w)  [a,1]*w';

%function handles for backprop:
bckp2 =@(y,t)  y-t;
bckp1 =@(w,z,d)(1-([z 1].^2)).*(w*d);

%training rounds
reps = 500;

%initiate weights (one added for bias...)
w1 = rand(3,M)-0.5;
w2 = rand(1,M+1)-0.5;
```

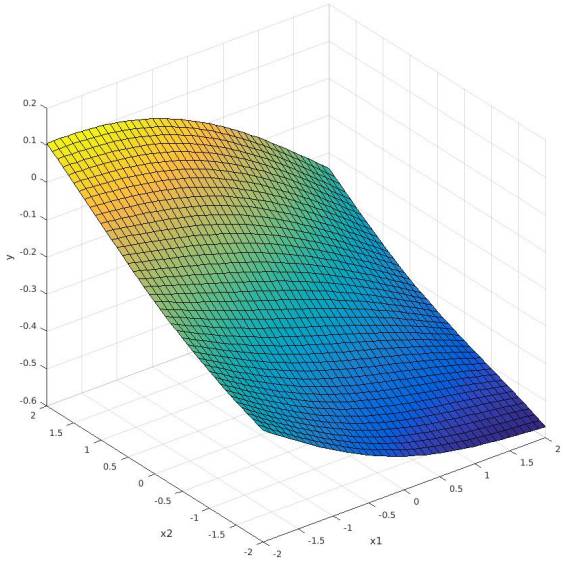


Figure 4: Output of neural network with random initial weights

Output of neural network is presented in figure 4. Neural network has 2 layers, 2 input nodes, 8 hidden nodes and 1 output node. Activation function for hidden nodes is $tanh(\cdot)$ whereas output node uses linear activation function. All weights are random values in $[-0.5, 0.5]$ interval.

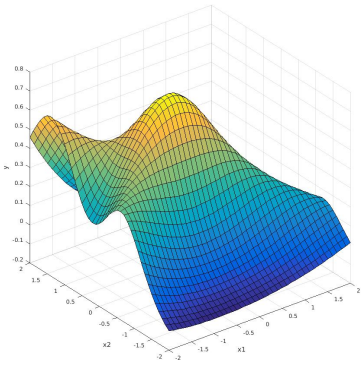
2.3 Training a network

On figure 5 we can see outputs of neural network after certain number of training cycles. On figure 4 we can see output given the random initial weights. After that all weights are updated using sequential error backpropagation procedure. Each cycle represents one iteration through whole data set. Weights are updated after each data point. On figures we can see that after 100 cycles plot starts to look like Gaussian density.T

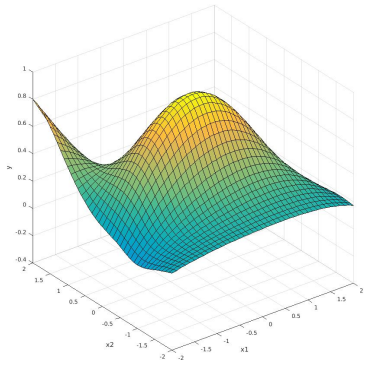
MATLAB code for iterative ANN training:

```
%training iterator:
for rep = 1:(reps*length(X(:)))
    %x,t points for this iteration (basically parsing through XY-grid
    %over and over)
    x=[X(mod(rep-1,length(X1)) +1) Y(mod(rep-1,length(X1)) +1)];
    t = tgt(x(1),x(2));

    %fwdprop:
    z = fwdp1(x,w1);
    y = fwdp2(z,w2);

    %backprop:
    d2 = bckp2(y,t) ;
    d1 = bckp1(w2,z,d2);
    d2s = [d2s,d2];
    %update weights:
    w1 = w1-(eta*repmat(d1(1:M),[3 1]).*repmat([x 1]',[1 M]));
    w2 = w2-(eta*repmat(d2,[1 M+1]).*repmat([z 1],[1 1]));

    %Update plot every Nth iteration:
    N=50;
    if ~mod(rep,length(X(:))*N)
        for i=1:length(X1)
            Z1(i)=fwdp2(fwdp1([X1(i) Y1(i)], w1),w2);
            %Z(i) = tgt(X1(i),Y1(i));
        end

        subplot(2,5,(rep/length(X(:)))/N)
```

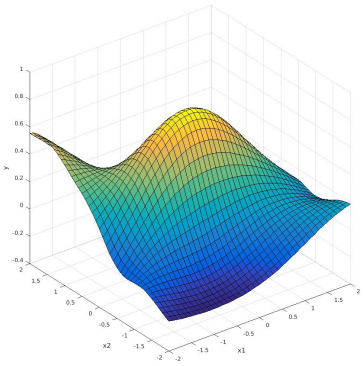
```
29          surf(X,Y,Z1);
30          title((rep/length(X(:))));
31          drawnow
32      end
33  end
```



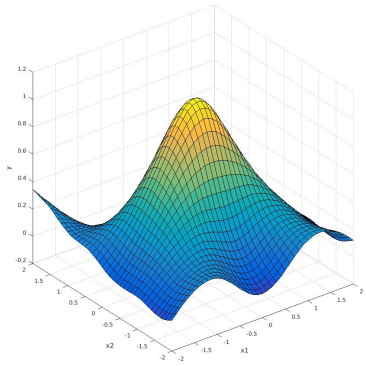
(a) Output after 20 cycles



(b) Output after 100 cycles



(c) Output after 200 cycles

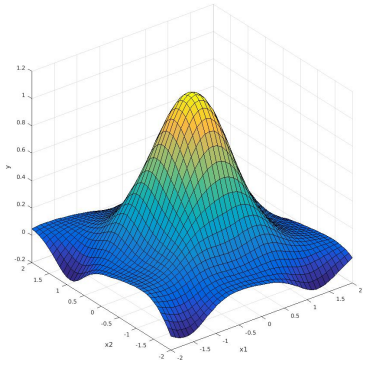


(d) Output after 500 cycles

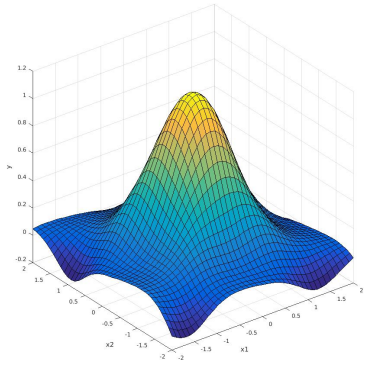
Figure 5: Output of neural network after 20, 100, 200 and 500 cycles with learning rate 0.1 and 8 units in hidden layer

2.4 A less patterned input

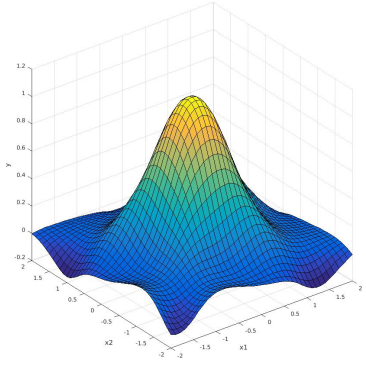
If we randomly permute our training input and output sets we can see that the convergence is much quicker. Even after 10 cycles plot resembles Gaussian density. That can be seen on figure 6. Randomized data set obviously leads to faster convergence. That is because if we have data set ordered in some way, weights will update slowly according to presented order. So, for example if we have ordered clusters of data where every cluster represents different class, after iterating through first cluster weights will be updated according to that cluster, than when we start iterating through second cluster weights will start to update according to that cluster etc. It is better to randomly pick data set points so our learning algorithm won't suffer from any effects of order in data.



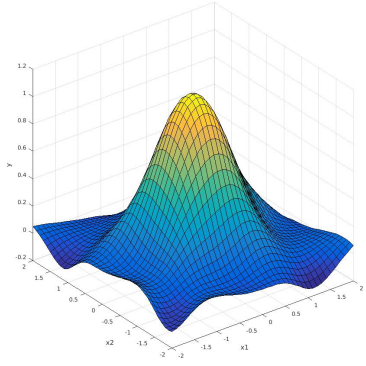
(a) Output after 10 cycles



(b) Output after 20 cycles



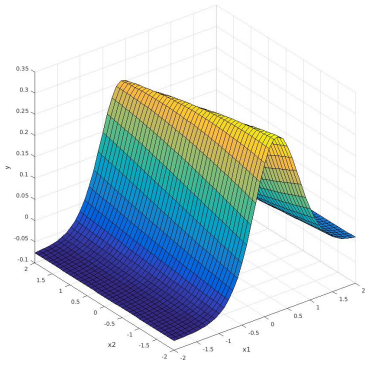
(c) Output after 50 cycles



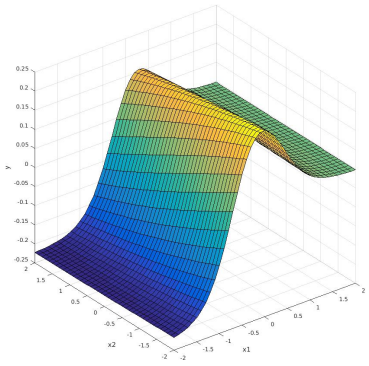
(d) Output after 100 cycles

Figure 6: Output of neural network after 10, 20, 50 and 100 cycles with learning rate 0.1 and 8 units in hidden layer

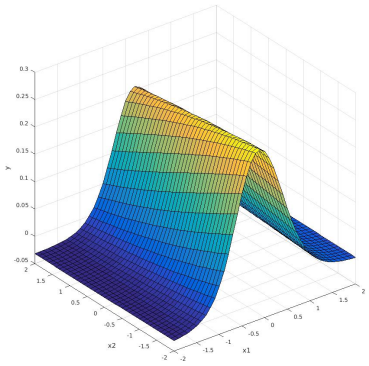
If we increase learning rate to 0.4 we can notice that algorithm doesn't converge at all as we can see on figure 7. On the other hand, if we decrease learning rate to 0.001 we can notice that algorithm converge but much slower as presented in figure 8.



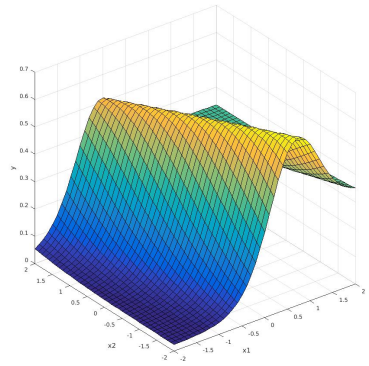
(a) Output after 10 cycles



(b) Output after 20 cycles



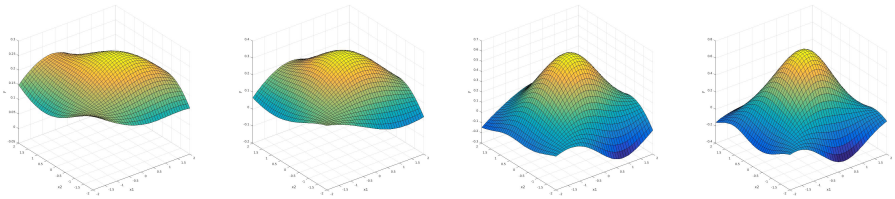
(c) Output after 50 cycles



(d) Output after 100 cycles

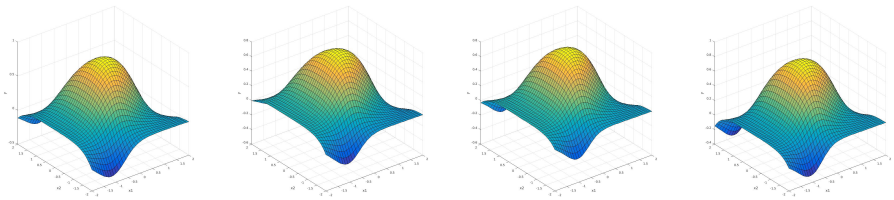
Figure 7: Output of neural network after 10, 20, 50 and 100 cycles with learning rate 0.4 and 8 units in hidden layer

We have also tried changing number of units in hidden layer in order to see its effect on learning. From figures 8 and 9 we can see plots for networks with 3 and 25 hidden units, respectively. It is obvious that network with more hidden units performs better in this case. But adding many units in hidden layer doesn't help that much, results are similar with 8, 15, 25 or more units. But reducing number of hidden units reduces network's learning capabilities and its expressiveness:



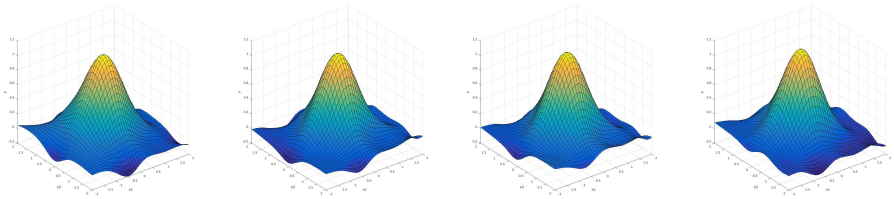
(a) Output after 10 cycles (b) Output after 20 cycles (c) Output after 50 cycles (d) Output after 100 cycles

Figure 8: Output of neural network after 10, 20, 50 and 100 cycles with learning rate 0.001 and 8 units in hidden layer



(e) Output after 10 cycles (f) Output after 20 cycles (g) Output after 50 cycles (h) Output after 100 cycles

Figure 8: Output of neural network after 10, 20, 50 and 100 cycles with learning rate 0.1 and 3 units in hidden layer



(a) Output after 10 cycles (b) Output after 20 cycles (c) Output after 50 cycles (d) Output after 100 cycles

Figure 9: Output of neural network after 10, 20, 50 and 100 cycles with learning rate 0.1 and 25 units in hidden layer

2.5

Required plot is shown in figure 10.

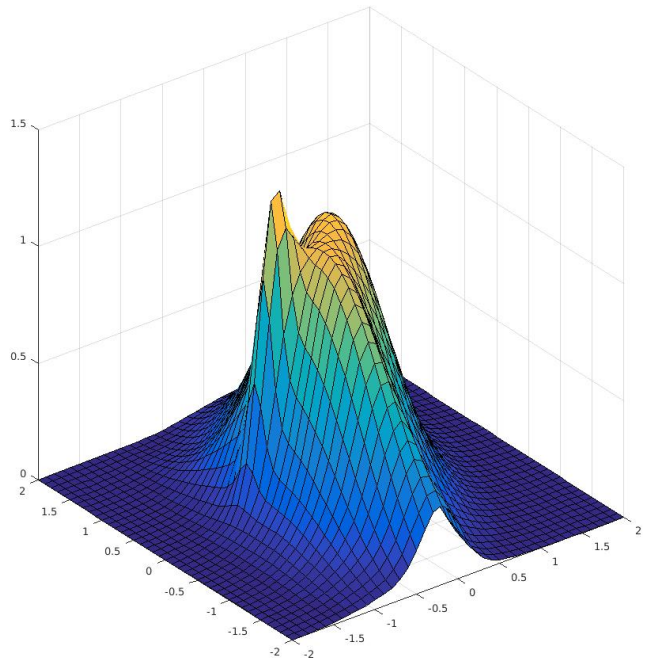


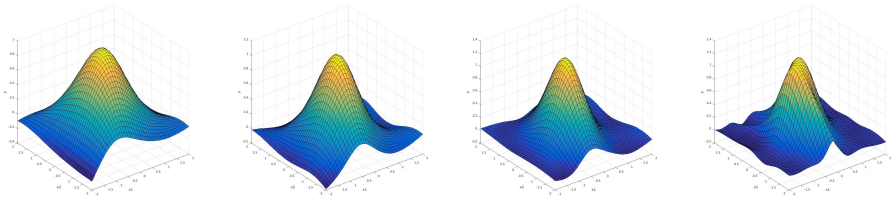
Figure 10: 2D plot of the target probability density function

2.6

We used network with 50 units in hidden layer and we set learning rate to 0.005 to train our neural network for data set given in previous task. On figure 11 we can see plots of our results after certain number of training cycles. If we compare our results with plot presented in 10 (which is actually a plot from given data set) we can notice one big difference: our plots are missing that "peak anomaly" that can be seen on figure 10.

2.7

Using *Netlab* toolbox we created network with 50 units in hidden layer, we trained our data set using scaled conjugate gradients. Final output is presented in figure 12. As we can see, our results (figure 11) are similar to this result, but this one looks slightly better at approximating function from given data set.



(a) Output after 20 cycles (b) Output after 40 cycles (c) Output after 100 cycles (d) Output after 2000 cycles

Figure 11: Output of neural network after 20, 40, 100 and 2000 cycles with learning rate 0.005 and 50 units in hidden layer

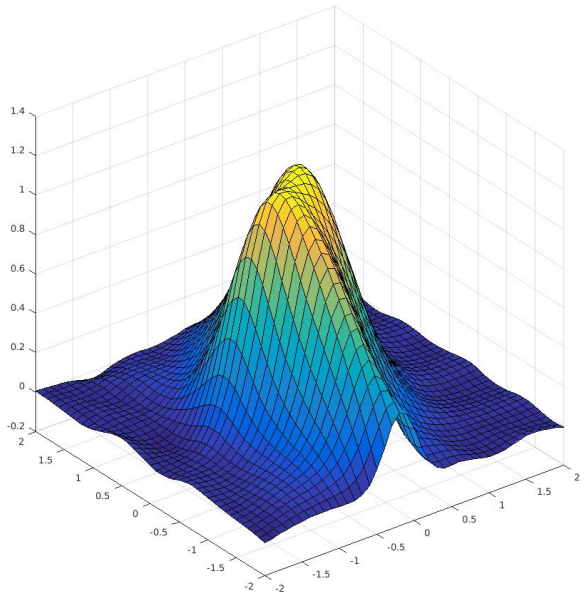


Figure 12: 2D plot of neural network output that was trained and implemented using *Netlab* toolbox

3 Exercise 3 - EM and doping

3.1

In figure 13 we can see histogram representing distribution of each variable (x_1, x_2, x_3, x_4). Figure 14 depicts a 3D plot of all 4 variables where dot color

stands for variable x_4. Based on that graph we made our prediction of data clusters (black ellipses on the graph). We think that there might be 4 different classes and that the "middle" ellipse represents athletes that have substance X in their blood because that cluster shows strong correlation between variables x_1 and x_2.

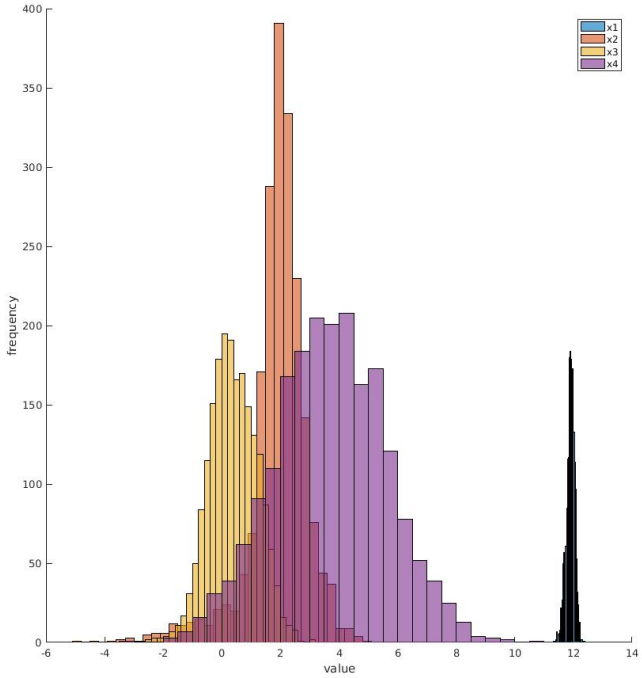


Figure 13: Histogram of given data (x_1, x_2, x_3, x_4)

3.2

EM algorithm:

- Initialize the means $\boldsymbol{\mu}_k$, covariances $\boldsymbol{\Sigma}_k$ and mixing coefficients π_k, and evaluate the initial value of the log likelihood.
- **E step.** Evaluate the responsibilities using the current parameter values

$$\gamma(z_{nk}) = \frac{\pi_k \mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)} \tag{15}$$

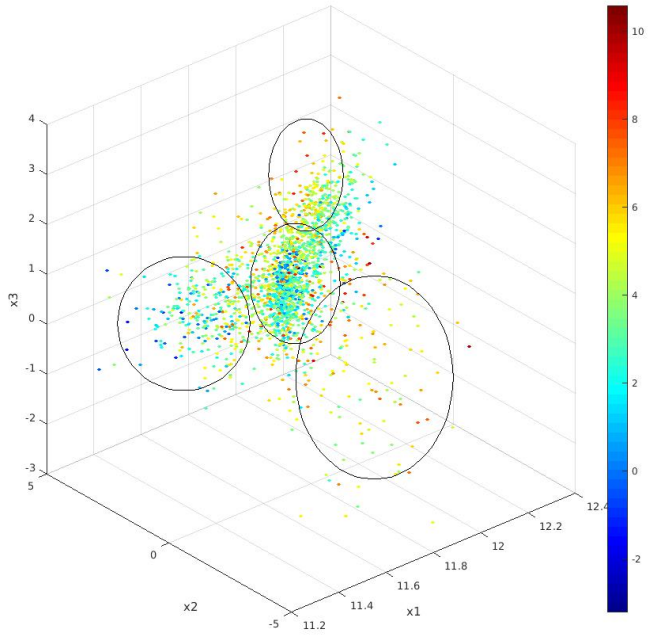


Figure 14: 3D scatter plot representing all 4 variables (x_1, x_2, x_3, x_4) of given data where color represents variable x_4

- **M step.** Re-estimate the parameters using the current responsibilities

$$\boldsymbol{\mu}_k^{new} = \frac{1}{N_k} \sum_{n=1}^{N} \gamma(z_{nk}) \boldsymbol{x}_n \tag{16}$$

$$\boldsymbol{\Sigma}_k^{new} = \frac{1}{N_k} \sum_{n=1}^{N} \gamma(z_{nk})(\boldsymbol{x}_n - \mu_k^{new})(\boldsymbol{x}_n - \mu_k^{new})^T \tag{17}$$

$$\pi_k^{new} = \frac{N_k}{N} \tag{18}$$

  where

$$N_k = \sum_{n=1}^{N} \gamma(z_{nk}) \tag{19}$$

- Evaluate the log likelihood

$$ln(p(\boldsymbol{X}|\boldsymbol{\mu}, \boldsymbol{\Sigma}, \boldsymbol{\pi})) = \sum_{n=1}^{N} ln(\sum_{k=1}^{K} \pi_k \mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)) \tag{20}$$

and check for convergence of either the parameters or the log likelihood. If the convergence criterion is not satisfied return to step 2.

Next we present our implementation of EM algorithm in MATLAB where initialization is done in the way described in this exercise.

```
X = load('a011_mixdata.txt', '-ASCII');
[N, D] = size(X);

%INITIALIZATION

%number of classes is set to 2
K = 2;
pi = [0.5 0.5]';
meanx1 = mean(X(:,1));
meanx2 = mean(X(:,2));
meanx3 = mean(X(:,3));
meanx4 = mean(X(:,4));
means = [meanx1 meanx2 meanx3 meanx4];

%columns represents variable from x1 to x4
%rows represents K different classes
mu = zeros(K,D);
%sigma(:,:,k) represents covariance matrix for k-th class
sigma = zeros(D, D, K);
for i=1:K
   mu(i,:) =  means + rand(1,4).*2-1;
   sigma(:,:,i) = diag(4.*rand(1,D)+2);
end

initLH = evalLogLH(X, mu, sigma, pi);
fprintf('Init log likelihood = %f\n', initLH);

maxIter = 100;
for i=1:maxIter
    %Expectation step
   resp = responsibility(X,mu,sigma,pi);

   %Maximization step
   Nk = sum(resp);
   for k=1:K
      mu(k,:) = (resp(:,k)'*X)./Nk(k);
   end

```

```
    for k=1:K
       temp = zeros(D,D);
       for n=1:N
           temp = temp + resp(n,k).*(X(n,:)-mu(k,:))'*(X(n,:)-mu(k,:));
       end
       sigma(:,:,k) = temp./Nk(k);
    end

    pi = Nk./N;

    LH = evalLogLH(X, mu, sigma, pi);
    fprintf('Step %d: log likelihood = %f\n', i, LH);
end
```

Plot routine that plots the x_1, x_2 coordinates of data points and color each data point according to the most probable component according to the mixture model is given by following MATLAB code:

```
%probs is a NxK matrix where N is number of data set points and K is
%number of classes. Every row of this matrix represents probabilities
%of assigning given point to class represented by column.
probs = zeros(N,K);
for k=1:K
    probs(:,k) = mvnpdf(X, mu(k,:), sigma(:,:,k));
end
[M, I] = max(probs');

%scale I
a = 0; b = 1;
mn = min(I); mx = max(I);

mycolormap = colormap('winter');
inp = 0:63;
d64 = inp./63;
I = ((b-a).*(I-mn))./(mx-mn) + a;
c = interp1(d64, mycolormap, I);

dotsize = 10;
sca = scatter(X(:,1), X(:,2), dotsize,c,'fill');
colorbar;
xlabel('x1'); ylabel('x2');
```

3.3

For K=2 log likelihood converges to -7058.126254 in 40-50 steps. On figure 15 we can see scatter plot of the x_1 and x_2 coordinates colored according to the most probable component. Even with different initializations we got very similar results for K=2.

Correlation coefficient is -0.053 for blue class on graph and -0.323 for green class on graph. This means there is no correlation in the blue class, and a medium negative correlation in the green class. Neither class shows the characteristic strong positive correlation between x_1 and x_2. Therefore it is safe to say that the data is not made up of only two Gaussians.

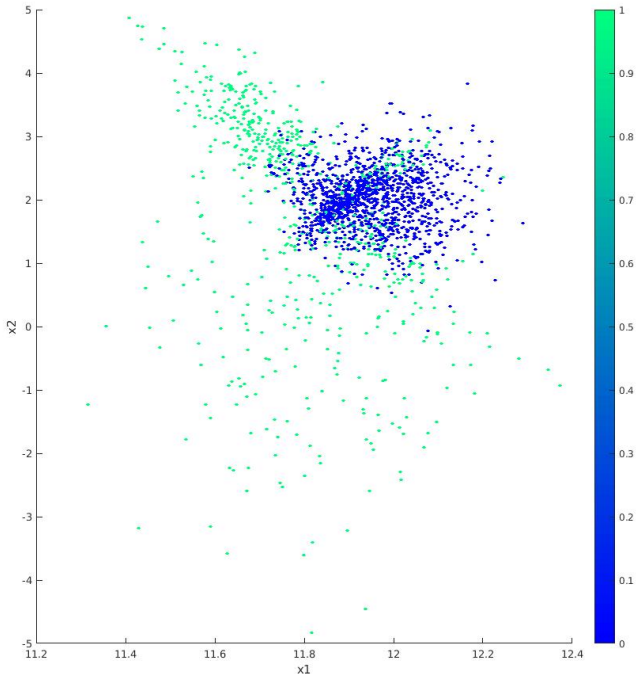


Figure 15: Scatter plot of data points colored according to most probable component for K=2

3.4

For K=3 log likelihood converges to -6551.130294. On figure 16 we can see scatter plot of the x_1 and x_2 coordinates colored according to the most probable component. Correlations for the green, blue and red classes are respectively: 0.081, -0.72 and -0.017. Still, there is no strong positive correlation in the components.

For K=4 log likelihood converges to -5989.752406. On figure 17 we can see scatter plot of the x_1 and x_2 coordinates colored according to the most probable component. Correlations for red, dark blue, light blue and yellow classes are respectively: 0.92, 0.036, -0.89 and -0.053. These show the strong positive correlation between x_1 and x_2 in the red class and the strong negative correlation that was hypothesized as a possible group for clean athletes in the light blue class. From the plots for various values of K we can assume that K = 4 is the most likely case. The portion of all data points in the red class is 0.1958. Therefore it is safe to say that the estimate of 1-in-5 of the users using doping is credible.

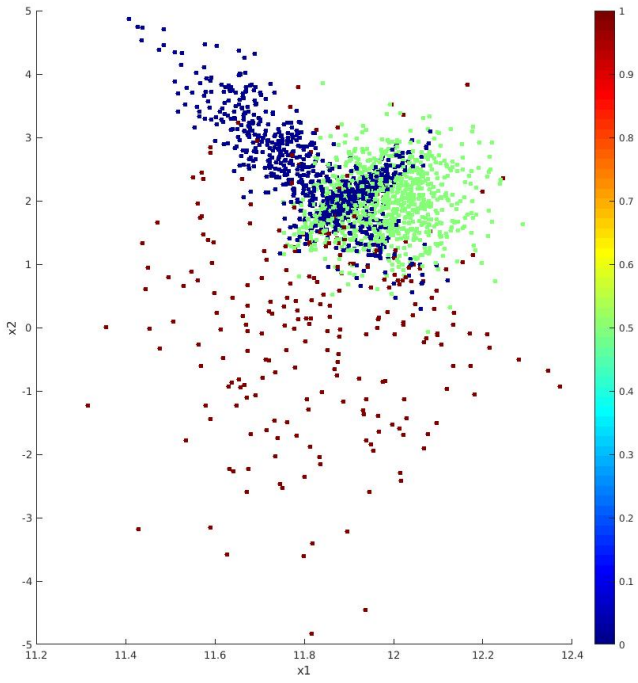


Figure 16: Scatter plot of data points colored according to most probable component for K=3

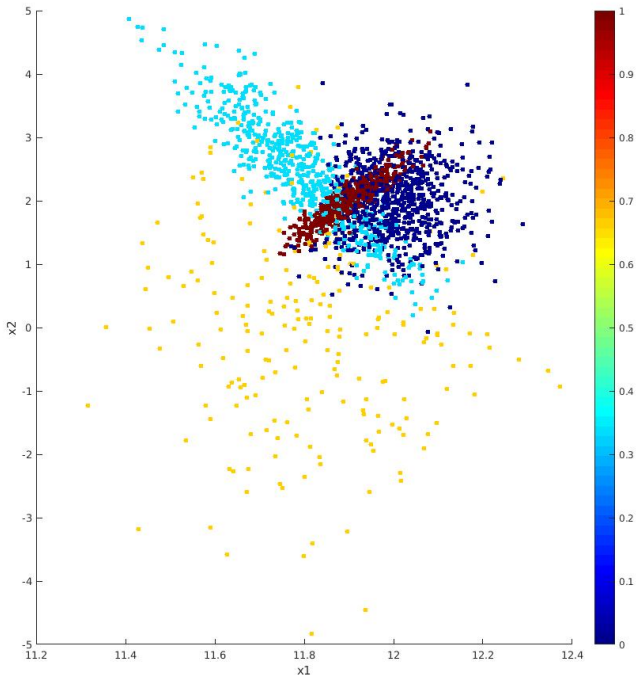


Figure 17: Scatter plot of data points colored according to most probable component for K=4

3.5

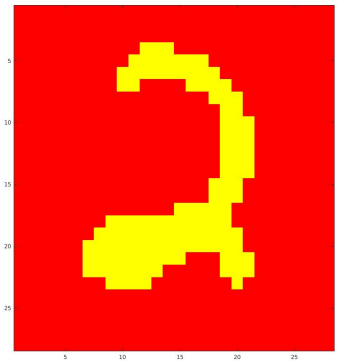
In matrix presented in expression 21 we can see probability of samples A, B, C and D belonging to certain classes. Rows represents samples A, B, C and D while columns represents probabilities of belonging to class 1 (dark blue), 2 (light blue), 3 (yellow) or 4 (red), respectively. From that matrix we can conclude that subject C took the drug X because sample C is most likely to belong to class 4 (red). Subject B tried to tamper with the test because probability of belonging to any class for that sample is 0 or very close to 0.

$$\begin{pmatrix} 0.1218 & 0.0857 & 0.0012 & 0.0001 \\ 0.0000 & 0.0000 & 0.0000 & 0.0000 \\ 0.0000 & 0.0000 & 0.0000 & 0.2484 \\ 0.1089 & 0.0000 & 0.0007 & 0.0000 \end{pmatrix} \tag{21}$$

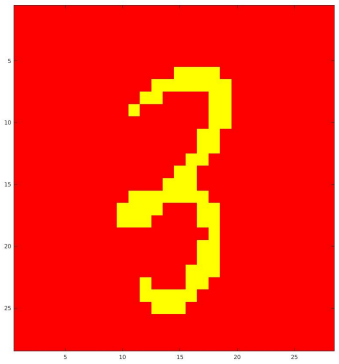
4 Exercise 4 - Handwritten digit recognition

4.1

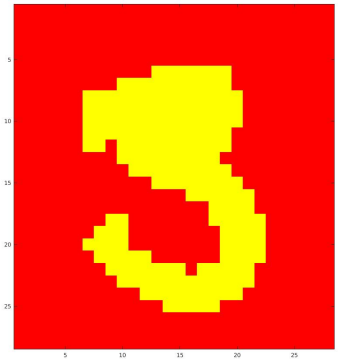
Some of the digits from data set are given in figure 18. We think it will be hard to classify them correctly since every number can be handwritten in many different ways. For example, in figure 18 we can see that number 3 is written in 2 completely different ways.



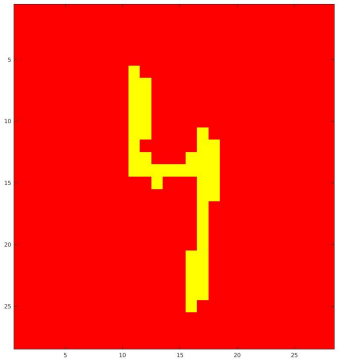
(a) Two



(b) Three



(c) Three



(d) Four

Figure 18: Examples of handwritten digits from data set

4.2

The algorithm is very similar to the algorithm that was used in the first exercise. The only difference being that a Bernoulli pdf is used instead of a Gaussian, and

the computation of Σ is omitted. Images are drawn after each iteration using following MATLAB snippet:

```
mudraw = mu.*255;
d = sqrt(D);
for k=1:K
   image(reshape(mudraw(k,:),d,d));
   colormap(gray(255));
   pause;
end
```

4.3

In figure 19 greyscale images of pixel means are presented. First row stands for means after first iteration and second row for means after 40 iterations. Algorithm converges quickly and from the images we can clearly see numbers 2, 3 and 4. Even with different initializations we got similar results and algorithm converged quickly.

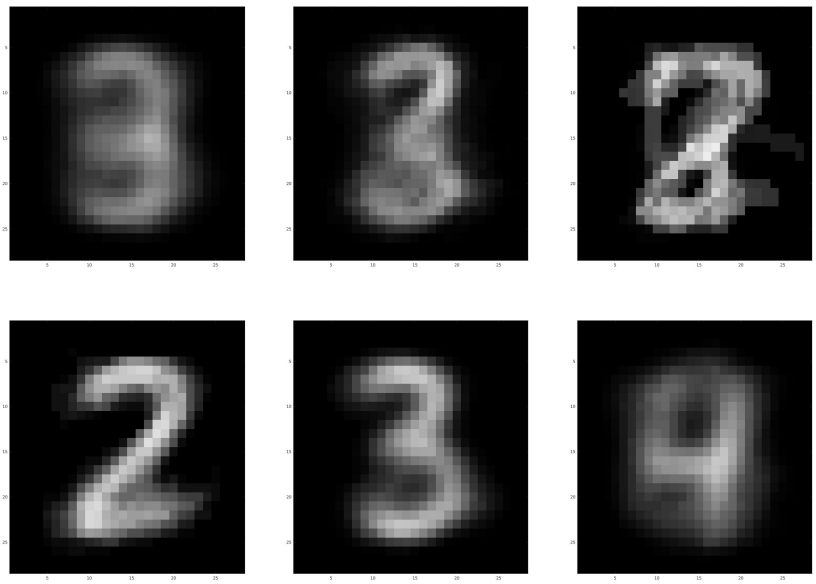


Figure 19: Greyscale images of pixels means where K = 3

4.4

In figure 20 we can see images of means with 2 classes and in figure 21 images of means with 4 classes. If we use $K > 3$ we get interesting results. Most of the time, we get cluster prototypes for different variations of the same digit, e.g. a curly 2 vs. a straight 2 or a closed 4 vs. an open 4 etc. For 2 classes we usually got results of some kind of hybrids between some digits.

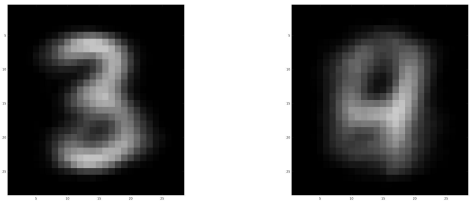


Figure 20: Greyscale images of pixels means where K = 2

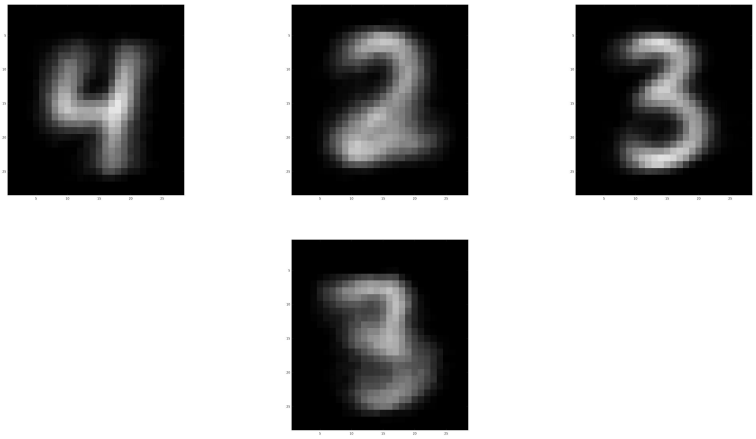


Figure 21: Greyscale images of pixels means where K = 4

Our algorithm correctly classified 91.75% of images. In figure 22 we can see one of the missclassified images. Our algorithm classified it as number 4 while it is actually number 3. Every human would say that it is number 3 but it isn't a "proper" shape of the number so it has been missclassified by algorithm. Based on that we can conclude that it is hard for algorithm to classify different shapes of particular numbers correctly.

If we initialize our algorithm with true mean values than it converges more quickly and there are less missclassified images.

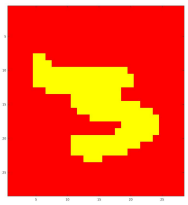


Figure 22: Missclassified number 3

4.5

We created a small dataset of 15 integers in the MNIST style and calculated its log likelihoods on our bernoulli class prototypes, which can be seen in figure 23. 7/15 (0.4667) of the images could be given the right label. Noticeably, the scale of the digits in the image is significant. Some of the digits that were drawn bigger and close to the boarder were missclassified with infinite p(x).

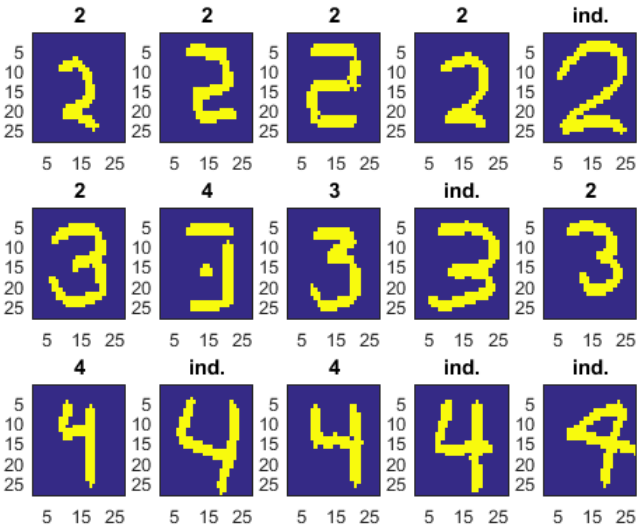


Figure 23: MAP- classifications of a self-created MNIST-style data set.