Assignment #5

This assignment is due on **May 28th one hour before class** via email to mailto:christian.wallraven+EMS2019@gmail.com.

If you are done with the assignment, make one zip-file of the `assignment5` directory and call this zip-file `STUDENTID1_STUDENTID2_STUDENTID3_A5.zip` (e.g.: `2016010000_2017010001_A5.zip` for a team consisting of two students or `2016010000_2017010001_2017010002_A5.zip` for a three-student team). The order of the IDs does not matter, but the correctness of the IDs does! **Please double-check that the name of the file is correct!!**

Please make sure to comment the code, so that I can understand what it does. Uncommented code will reduce your points!

ALSO: I can also surf on the internet for code. Downloading and copying and pasting other peoples' code is plagiarism and will NOT be tolerated. Please also clearly state whom you worked with (if applicable)!

Part1 Gradient descent (30 points):

Your task is to implement and test the gradient descent algorithm that looks for the minimum of a function $f:\Re^n \rightarrow \Re$, given its gradient $\vec{g} = \nabla f = \begin{pmatrix} \partial f/\partial x_1 \\ \partial f/\partial x_2 \\ \dots \\ \partial f/\partial x_n \end{pmatrix}$. Note that the function f takes a vector as an input and returns a number as an output, whereas the gradient takes a vector as input and returns a vector as output! Please take a look at the lecture slides on gradient descent again to familiarize yourself with the idea and the algorithm.

Note that gradient descent is an iterative algorithm. You need to choose a starting value for optimization x_0. Then you're trying to shoot in the direction of the current gradient of the function. That is, your next x-value is given by

$$x_k = x_{k-1} - \lambda \nabla f(x_{k-1})$$

Obviously, you need to choose λ carefully, otherwise strange things might happen. In addition, the algorithm needs a stopping criterion. We will be supplying two criteria here. In addition, the function $f:\Re^2 \rightarrow \Re$, so that the gradient will contain two parts. Hence, the function definition in Matlab should look like the following:

```
function [xoptimal,foptimal,niterations] = gradient_descent
(f,g1,g2,xstart,lambda,tolerance,maxiterations)
```

where `f` is a **function handle** to the definition of the function, `g1,g2` are **function handles** to the definition of the partial derivatives of the function f, `xstart` is an array of starting values for the optimization (you will need 2 coordinates of course), `lambda` is the update rate in the gradient descent step, `tolerance` is the value of the **squared norm of the gradient** at which you accept that the minimum is found (it should be $||g||^2_{L2} = 0$, but for numerical stability, usually a small, non-zero value is chosen), and `maxiterations` is the maximum number of iterations that you allow to take place.

Now let's go back to our function definition. The function should return `xoptimal`, which is the optimal point (an array of course), `foptimal`, which is the function value at `xoptimal`, and `niterations`, which is the number of iterations the algorithm made to reach criterion.

Now we're going to test your algorithm.

Use

```
f = @(x1,x2) x1.^2 + x1.*cos(x1.*x2/3) + 3*x2.^2;

xstart = [10 10]'; lambda = 0.03; tolerance = 1e-7, maxiter =
1000;
```

Now define the suitable gradient function. **Do NOT use the numerical gradient, but use your brain to actually derive the function!!**

This gives you two partial derivatives that you will use in your gradient descent function:

```
g1 = @(x1,x2) …

g2 = @(x1,x2) …
```

Write a script `gradientTestFunction.m` that calls the function and outputs the results for `[xoptimal,foptimal,niterations]`

We chose for updating each step `lambda = 0.03`. If your step size is too large, the optimization will not converge. What is the value of `lambda` **up to two digits** that will result in non-convergence? Try this out yourself and insert the value of `lambda` into the script.

Similarly, if you make larger steps, you could reach the minimum faster. What is the value of lambda that has the **minimum number** of steps in order to reach the minimum point? Try this out yourself and insert the value of `lambda` into the script.

Part2 Fix me (50 points – challenging problem!!):

I have uploaded a set of functions that perform standard backpropagation learning of a one-hidden-layer neural network. The files should be self-explanatory and commented.

Importantly, there is a file called `feed_forward_backprop_faulty_test.m`, which contains a simple test of the feed forward and backpropagation passes through the network. Both of these functions contain errors and their functionality should be extended.

a) check the two files and **fix the errors** – in doing so, pay attention to the dimensionality of the arrays for input and output, and for the backpropagation in particular, check that the correct, connected downstream neurons are accessed and summed over. This is probably the most difficult part of the assignment – please try it first by checking the class notes and resources on implementing the backpropagation algorithm - if you are truly stuck, I will send you some sample output data of the working version so you can debug (upon request only).

b) implement all TODOs in the files, extending the functionality with error checking and additional input so that the functions can make use of different activation functions; for the latter simply re-use the file `activation.m` from class and add another called `activation_der.m` which implements their derivatives!

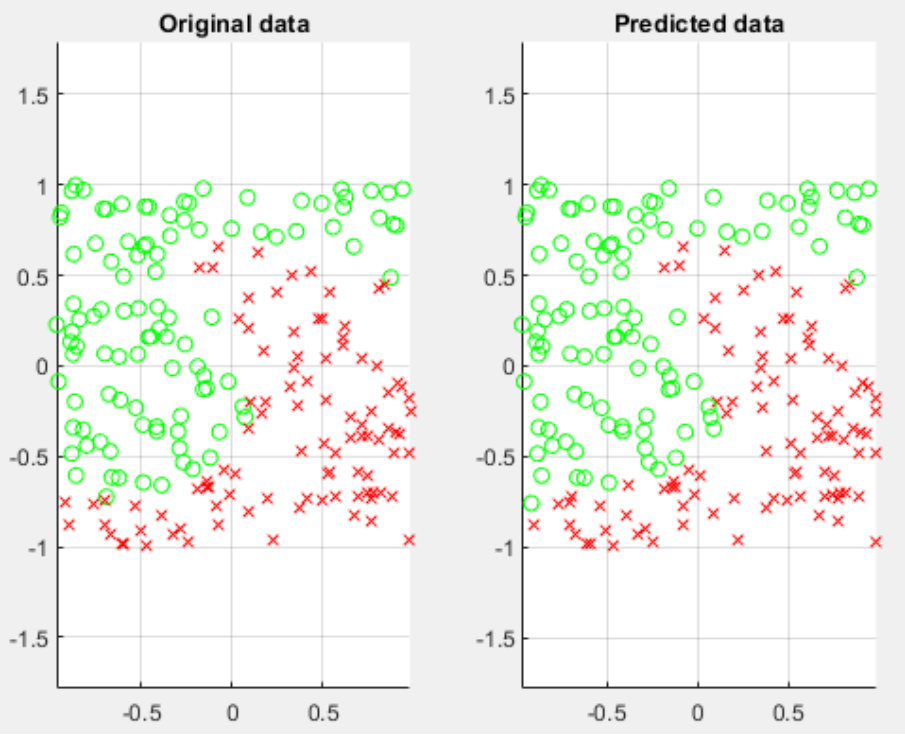
c) Implement a simple test called `testOneLayer.m`, which loads the data from below, changes the class-labels to 0 and 1, and learns the two classes using a one-hidden-layer neural network with the logistic activation function.

The dimensionality of the input layer should be clear, the dimensionality of the hidden layer is **up to you**, and the dimensionality of the output layer should be 1.

Train for 10000 epochs with a learning rate of .01, and use stochastic gradient descent, selecting a random point from the 200 datapoints in each epoch [see the class example for doing this].

Since the loss-function we have been using is ||targets – predictions||^2, our end-result will not be classification, but regression of sorts, so the network tries to make the predictions like 0 and 1 for each point. However, this is of course not necessary for classification itself. So, in order to classify the points, you can simply say that class 1 is predictions>.5 and class 0 is predictions <.5

Plot the original points as a scatter plot with class 0 in red, and class 1 in green. In a second plot, plot the predicted classes in similar color like so [as you can see, I can solve this reasonably well with my network 😊]



% two wave classes, 200 points, change labels to 1, 0!

```
0.342250    0.183500    2
-0.959500   -0.085500   1
-0.626500   -0.617500   1
-0.183000   0.539000    2
0.418500    -0.736000   2
-0.265250   0.805250    1
-0.247750   -0.972500   2
-0.597000   0.303500    1
-0.449000   0.162250    1
0.820250    -0.703000   2
-0.016000   -0.709250   2
-0.523250   0.610750    1
-0.462500   0.159500    1
0.171250    0.084500    2
0.160000    -0.264250   2
-0.618500   -0.187750   1
0.321500    -0.111000   2
-0.460000   0.879250    1
-0.414750   0.618000    1
-0.952250   0.823000    1
0.783250    -0.701750   2
0.382000    -0.468250   2
0.384250    0.913000    1
-0.869750   -0.196750   1
0.160000    0.741000    1
-0.695500   -0.722250   1
-0.270000   -0.531750   1
0.770250    0.969000    1
0.088250    -0.809250   2
-0.859250   0.108500    1
0.701000    -0.719000   2
0.980250    -0.476250   2
-0.729000   -0.418250   1
-0.350000   0.267500    1
-0.239000   0.901000    1
-0.341500   0.718750    1
0.771500    -0.727750   2
-0.159500   -0.128500   1
0.864500    0.955750    1
0.224250    -0.965000   2
-0.069500   -0.363750   1
-0.080250   0.656750    2
0.187250    -0.194750   2
-0.121000   -0.506500   1
0.910250    0.776500    1
0.867750    -0.151250   2
-0.517750   0.318500    1
-0.506000   -0.912000   2
-0.138250   -0.634750   2
-0.362500   0.161250    1
0.059250    -0.224250   1
-0.877750   0.343000    1
0.718000    -0.389750   2
-0.021750   -0.086000   1
-0.609250   0.895500    1
-0.866000   0.998500    1
-0.389250   -0.658250   1
0.885250    -0.719250   2
0.343000    -0.013000   2
0.980250    -0.182750   2
-0.083500   -0.770000   2
0.672500    0.661000    1
-0.970250   0.228750    1
-0.423500   0.520750    1
-0.079250   -0.876000   2
0.675000    -0.826750   2
-0.704750   0.068750    1
```

```
-0.681250   -0.155500   1
-0.886500   -0.483750   1
0.569500    -0.720500   2
-0.182250   0.753750    1
-0.283500   -0.275750   1
-0.893500   0.132750    1
-0.673750   -0.470750   1
-0.160000   0.980500    1
0.656250    -0.396250   2
-0.824250   -0.352250   1
-0.217500   -0.569000   1
-0.473750   0.673500    1
-0.713000   0.868250    1
0.431000    0.524000    2
0.841000    0.448250    2
0.941000    -0.113250   2
0.534250    -0.597750   2
0.605750    0.977250    1
-0.705750   -0.879000   2
-0.344500   0.833000    1
-0.766000   0.275250    1
-0.521500   0.065750    1
-0.724250   0.312000    1
0.978500    -0.967000   2
-0.103750   0.548250    2
0.914250    -0.094000   2
-0.668500   0.577000    1
0.810000    -0.404500   2
0.091000    0.208250    2
-0.468500   -0.988250   2
-0.879500   -0.338750   1
-0.876500   0.618750    1
-0.539750   -0.773000   2
-0.002250   0.759750    1
-0.260250   0.121250    1
0.071250    -0.283000   1
0.355000    0.744500    1
0.944500    0.978000    1
-0.328500   -0.930250   2
0.713750    -0.326000   2
0.095500    0.376750    2
-0.487500   0.662500    1
0.359000    0.047750    2
0.375750    -0.780000   2
0.860250    -0.349750   2
0.518750    -0.184250   2
-0.594500   -0.980500   2
0.627250    0.934000    1
-0.804250   -0.439000   1
0.249500    0.412500    2
-0.601500   0.495250    1
-0.112250   0.270750    1
-0.845750   0.257000    1
0.878250    0.488750    1
0.357750    -0.224750   2
0.658000    -0.286500   2
-0.944500   0.847000    1
-0.491000   -0.643500   1
-0.188750   -0.676750   2
-0.625250   0.052000    1
-0.127250   -0.665500   2
-0.156000   -0.054250   1
0.493500    -0.739250   2
0.891750    0.783500    1
-0.755000   0.677250    1
0.476750    0.265750    2
-0.330250   -0.011750   1
-0.416000   -0.359750   1
```

-0.886500	0.968500	1
-0.190500	-0.002250	1
0.085500	0.932250	1
-0.920500	-0.756750	2
0.984500	-0.252500	2
-0.402500	0.209000	1
0.540500	-0.584250	2
0.615000	0.878750	1
0.813250	0.433000	2
-0.884500	0.192000	1
0.088000	-0.345750	2
0.014000	-0.601000	2
-0.481000	0.880250	1
-0.048250	-0.570000	2
-0.141000	-0.122250	1
-0.704250	-0.746750	2
-0.880000	0.066500	1
-0.575000	0.687000	1
0.511750	-0.424750	2
0.903250	-0.364000	2
0.495500	0.899000	1
0.629000	0.221250	2
-0.695000	0.867250	1
0.559000	0.768000	1
0.771750	-0.852250	2
-0.902000	-0.873000	2
0.799250	0.001000	2
0.746750	-0.705250	2
0.335000	0.505250	2
0.103250	-0.201750	2
-0.822250	0.972250	1
0.815000	0.818500	1
-0.532500	-0.230500	1
0.567250	-0.481500	2
0.199500	-0.729250	2
-0.290500	-0.454000	1
0.609500	0.152250	2
-0.391000	-0.823750	2
0.770250	-0.248250	2
-0.672750	-0.931000	2
0.245250	0.713750	1
-0.288000	-0.896250	2
0.143500	0.632250	2
0.498250	0.265250	2
0.416250	-0.080250	2
0.515500	0.043750	2
0.744250	-0.389000	2
-0.411000	0.325750	1
-0.144250	-0.673000	2
-0.765500	-0.763250	2
0.618500	0.116000	2
-0.264500	0.908250	1
0.752750	-0.610000	2
-0.298750	-0.361750	1
-0.493500	-0.325000	1
0.696250	-0.587250	2
-0.863000	-0.604250	1
0.034250	0.257250	2
0.543500	0.411250	2
-0.665750	-0.614250	1
0.892250	-0.478250	2
0.930000	-0.378750	2
-0.417750	-0.335000	1
0.723000	0.046500	2
-0.609000	-0.979750	2