Assignment 2 TBMI26
Linus Mellberg (linme560) Lukas Gillsjö (lukgi451)
January 30, 2013

Backpropagation Network

The Data Sets

Question 1 If the available data is split into training data and validation data, it is possible to validate the classifier. If all available data is used for training the classifier might be very good at exactly that data, but bad at classifying new data. The validation data makes it possible to test how the classifier performs on data it hasn't been trained on. The performance of the classifier on the validation data is likely to reflect how the classifier performs on new data.

Question 2 The first data set consist of two separated cluster and a linear classifier can probably be used. The second data set consist of a cluster of one class with the other class surrounding it like a semicircle on one side. In this case a linear classifier can't be used. There is no straight line that can separate the clusters. The third set consists of three classes two semicircle-shaped ones and on cluster. This set also needs a non linear classifier, since there are three classes this set needs a classifier that handles three classes.

Question 3 The label matrix is a matrix where the columns are test cases, and the rows are classes. There are one row for each class. The class that the particular test case belong to has 0.99 as a value, and the other rows has -0.99. We don't really know why the data is between these values. We have been discussing it, but haven't found any real answer. One thing we have been discussing is that it may make it easier for the weights between the hidden layer and the linear classifiers, as the output from the hidden layer is limited by the tanh function we use. Say that we instead used values from 100 to -100. This means, that for the derivative of the cost function to be 0, the weights between the hidden layer and the linear classifiers needs to be big for the term U-D to be 0.
This also means that the gradient will be bigger, as the changes needed to the weights is bigger. We tested this by increasing the interval of the label matrices and lowering the learning rate. The result we got was the same as before. But if we kept the same learning rate as when the labels were in the interval 0.99 to -0.99, the algorithm didn't work.
So our answer is that the interval is needed so that the changes needed on the weights isn't that large, i.e. the gradient won't be very large.

Implementation

Question 1 High values in the input will generate high values in the hidden layers stepfunction (in this case tanh). The derivative of tanh is almost zero for high input values. This is not good since this derivative is what is used to find good weight values.

Question 2 We transformed the training data before using it. This means that the network we have trained is trained for the intervals that the training data is in. So if we apply this network to the test and evaluation data, it will not work, as that data has not been transformed the same way the training data has. That is why we do the transformation of those with the mean and variance from the training data.

Question 3 The learning rate decides how big steps we take in the direction of the gradient each iteration. There are several problems with a constant learning rate. If the learning rate is too big, we may miss local minimas totally by jumping over them. If the learning rate is too small, we may get stuck in the first local minima and not find anything more, we may even take such small steps that we won't find any minima at all.
To decide how big a step will be, we multiply the learning rate with the gradient. So the learning rate decides how big of a step we will take in the direction of the gradient.

Question 4 We need extra dimensions for the constant that decides the hyperplanes distance from origo. We need a variable that is not influenced by the input values in any way. These added weights are called the bias weight.

The Training

1.
How do you define "Best Performance"?
Best performance for us means the most correctly classified tests. The test data is more important than the training data, and if we get good results on the test and training data, we think the generalization is good enough.
What is the best performance?
With a setting of 1000 epochs, 300 hidden neurons and 0.1 learning rate we got 99.7% of the training data and 99.6% of the test data on problem X3. That was the best performance we got. The least important variable was the hidden neurons at this stage. We could lower the amount to 200, and still get results that were close to this. The other variables influenced more.
Is this expected? Why (not)?
Yes, this was expected. If the training data is good, the amount of errors in both should be about the same. More often, there will be more errors on the test data.
When we get "best performance" on the test data we got for the lab a few of the samples are often wrongly classified. This is because the different classes often are not perfectly separated there are samples from one class which are very close to samples in a different class.

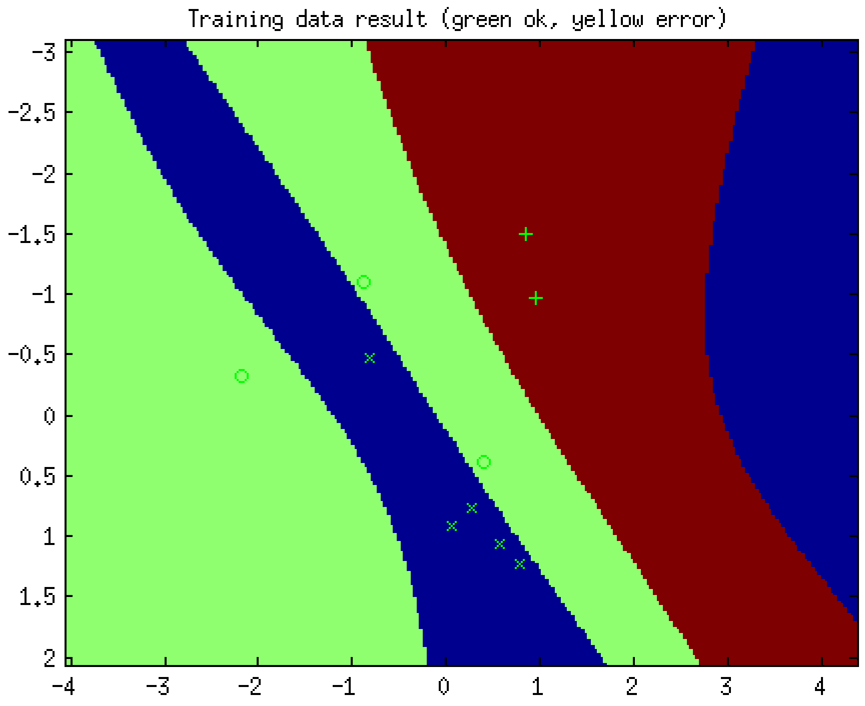
2.
What is the training performance?

Training performance is how many of the samples in the training data set that are correctly classified by the trained classifier. This should usually be higher than the number of correctly classified samples of the test data. This can happens for mainly two reasons. The training data can be bad and not represent the test data in a good way. The other reason is overtraining. Data is often not perfect, a few samples may be wrongly classified and there might not even be possible to have regions that clearly separates classes. When this happens an overtrained classifier will find regions that are good for the training data but not for general data. For example there might be "classification islands" around outliers that has features that should correspond to other classes in the general case. This will make the performance worse for new data as data that appears in these island should probably be classified differently.
When only a few training data samples are used the training performance often zero wrongly classified samples. When the classifier then is used on the test data the number of bad classifiers often is large. There is to little data to describe the classes and the classifier can't find good decision regiions. When about 60 samples are the number of bad classifications for the test data gets better. Now there is enough training data to represent the whole data set.

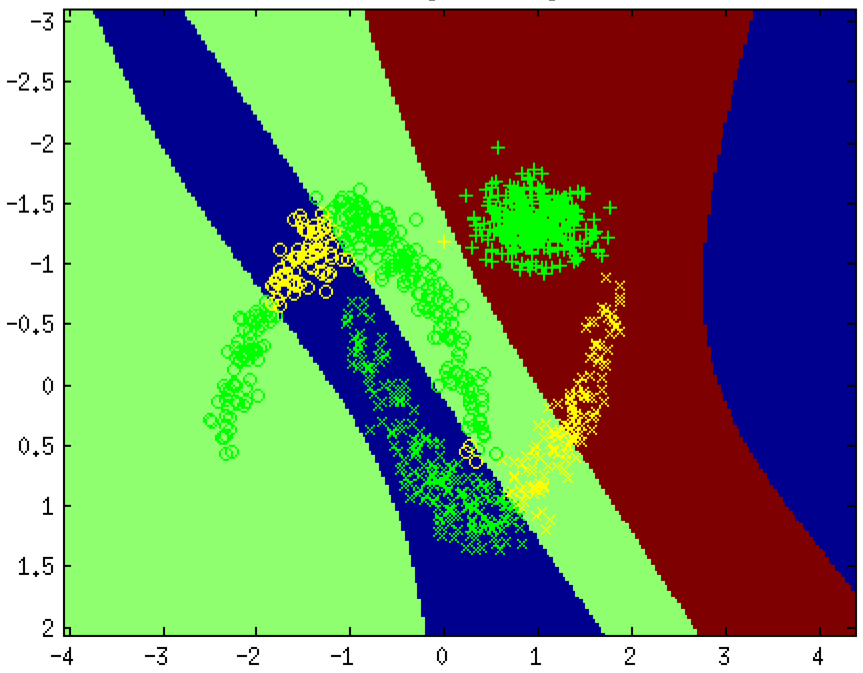
What do you call this kind of phenomenon?

Too little training data.

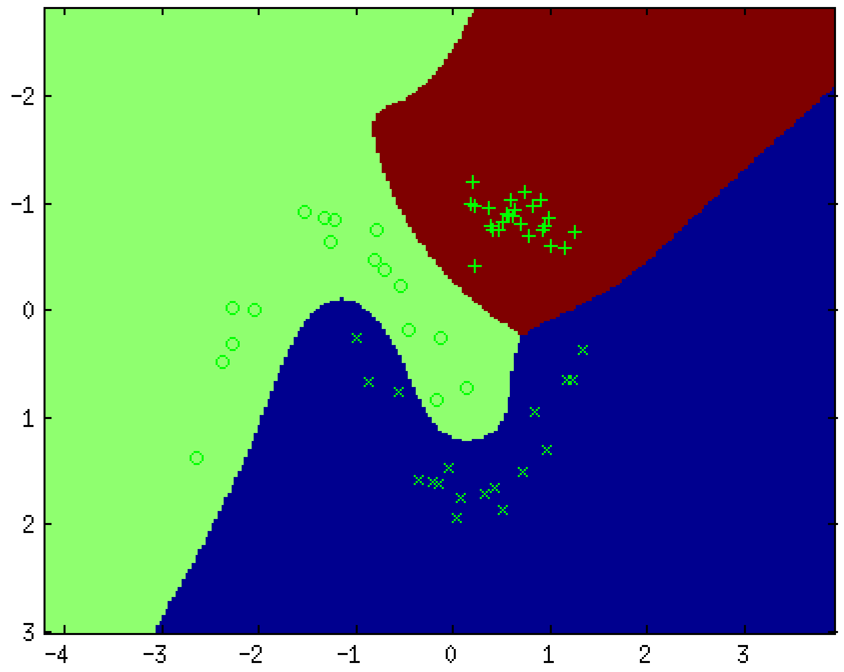
Include relevant training plots.

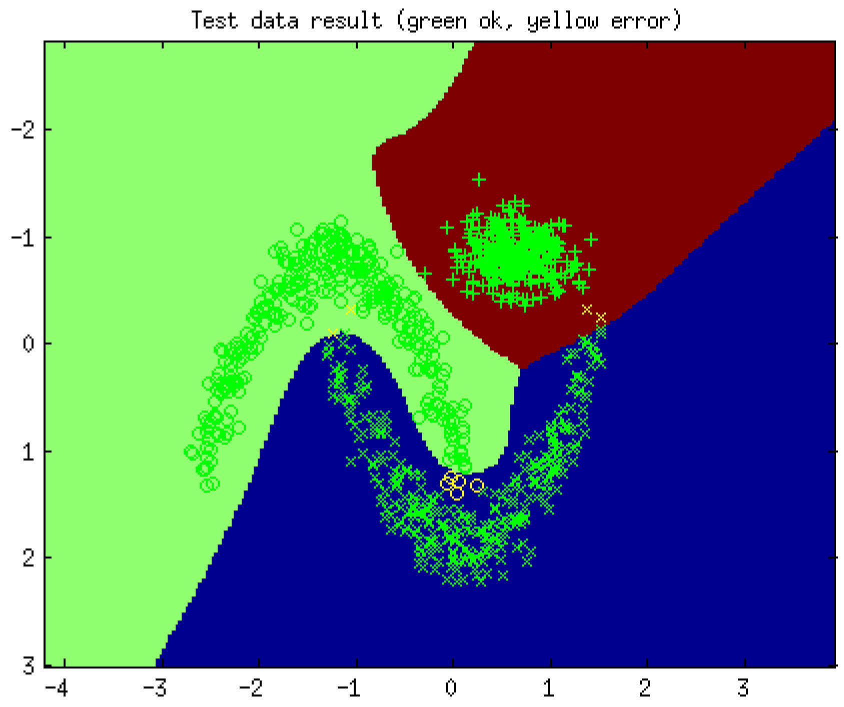


Test data result (green ok, yellow error)



Training data result (green ok, yellow error)





Optical Recognition of Handwritten Digits

Describe the data.
The data, in the form we have it in matlab, is a matrix where the columns are test data, and the rows are pixel values. Each test data has 64 pixel values between 0-16. These values represent how many white pixels there were in a grid of 4x4. So the original bitmaps was 32x32, but the one we have is a 8x8 one.
What is the training performance? Relate to the previous results.
The best we got was 98 % correct on the training data. This is bad compared to the results on the other problems.
The data has already been preprocessed, in what way?
As said earlier, the starting bitmaps was 32x32. The bitmap was then divided into 4x4 blocks, non-overlapping. The amount of white pixels were counted in each block, which gave a value between 0 and 16. These values were then entered into a 8x8 bitmap, which is the one we operate on. In our case, this bitmap is represented as an array of 64 values.
The preprocessing has several benefits both computational and classification wise. Why do we get these effects from this type of preprocessing?
It's easier to handle 64 input values than 1024. It decreases the dimensionality. Also, a distortion will not be as noticeable, as it will only change a few input variables, and probably not in any big ways. For example, a distortion inside a block will only change one variable, and not 16. This also means that all combinations of values inside a block that gives the same sum will be considered the same, but that is a problem we will have to live with.

You might also have noticed that the data normalization is turned off for this task, why?
The data is already normalized to be between the values 0 to 16, so no further normalization is needed.
Try running the same settings several times, how important is the initialization?
Initialization is not that important in our case. We still had a value around 98 % , it changed with about 0.2 percentage points with different initializations.

Support Vector Machines

A large margin classifier is one that tries to put a hyperplane between the classes and maximise the margin from this hyperplane to the closest data. This is good, as a larger margin usually means a lower generalization error. A support vector is an input data that lies on the border on the margin. These are enough to describe the position of the hyperplane.
The main weakness of the SVM is that it wants to totally separate the training data into two classes. This means, that if one input that is of class 1 is in the cluster that should be classified as class 2, the SVM will have problems as it cannot define a margin that doesn't contain any data points. So how can we solve the problem that SVM wants to divide the problem set cleanly? We can introduce the concept of a soft margin. This margin will split the problem set as cleanly as possible while still maintaining a margin as large as possible. We introduce a variable for each input data, *eta*, that will represent how misclassified that input is. We also introduce a punishment variable, *C*. The cost function of the SVM is then expanded by adding the sum of all *eta* multiplied with *C.* So we will also try to minimize this error. A problem with this approach is that the *C* variable is user supplied, which takes away from the ease of use of SVM.
SVM1 is the standard SVM, and SVM2 is one with soft margin. So a case where SVM1 will fail but not SVM2 is when we a clear partitioning of the data set into two classes, but one training point is mislabeled. So a point is of class 1, but is in the cluster of class 2.
We can use a kernel trick to make the SVM non-linear. This is a transformation of the feature space by using non-linear kernel functions.
One advantage of backprop over SVM is that it handles mislabeled training points much better. There is no need to adjust the algorithm to take care of them.
One advantage of SVM over backprop is that the solution is global and unique. There exists only one hyperplane that maximises the margin.