Report exercise sessions

Lode Bisschops

May 2022

1 Session 1

1.1 Function approximation: comparison of various algorithms

The following comparisons are done using a network with 1 hidden layer.

1.1.1 How does gradient descent perform compared to other algorithms?

None of the algorithms fit the data well with 5 neurons in the hidden layer.
Using 50 nuerons traincgf, traincgp, trainbfg, trainlm approximated the function well. Look at figure 1 for a visual representation of the function approximation using a different number of neurons.

Speed seems to be very similar between all the algorithms, as can be seen in table 1.

Accuracy traincgf, traincgp, trainbfg, trainlm far outperform regular gradient ascent, as can be seen in the charts of figure 1.
since the mean squared error - *the mean of the squared difference between the expected point and the actual result* is significantly lower in them.

Table 1: Time to train with 50 hidden layers

name	5 epochs	50 epochs	1000 epochs	# epochs
traingd	0.124	0.275	2.441	1000
traingda	0.085	0.219	0.396	126
traingdm	0.073	0.13	2.028	977
traingdx	0.071	0.168	0.446	151
traincgf	0.09	0.317	0.381	56
traincgp	0.086	0.304	0.368	56
trainbfg	0.111	0.444	0.848	107
trainlm	0.094	0.193	0.268	19

1.1.2 Learning from noisy data: generalization. Compare the performance of the network with noiseless data.

Time: Looking at table 2 , When working with noisy data the training of plain gradient descent doesn't stop, cause the validation criterion are never met.

accuracy Again the other algorithms trump gradient ascent, they fit the function significantly better, they can get most/all of the curves in the sine function after 50 iterations while the gradient ascent algorithms are far away from the target.

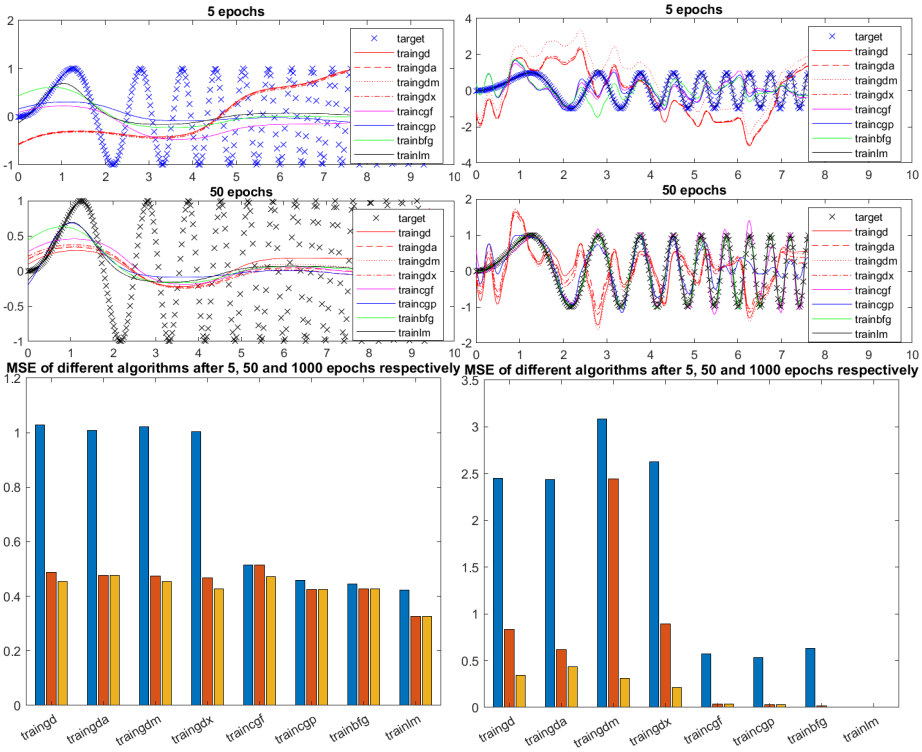

Figure 1: Comparison in fit of different algorithms. 5 hidden layers on the left, 50 hidden layers on the right.

Table 2: Time to train with noise.

name	5 epochs	50 epochs	1000 epochs	# epochs
traingd	0.081	0.229	3.045	1000
traingda	0.064	0.236	0.422	101
traingdm	0.057	0.153	2.862	972
traingdx	0.058	0.199	0.278	54
traincgf	0.067	0.172	0.471	76
traincgp	0.065	0.311	0.394	55
trainbfg	0.094	0.441	0.641	52
trainlm	0.069	0.159	0.248	17

1.1.3 Personal Regression example

Using the functions from the previous exercise, I analyzed their performance for this task. I used 1 hidden layer and analyzed the performance on the validation set for a different amount of neurons.
This can be seen in figure 3 Each model is trained a maximum of 1000 epochs.

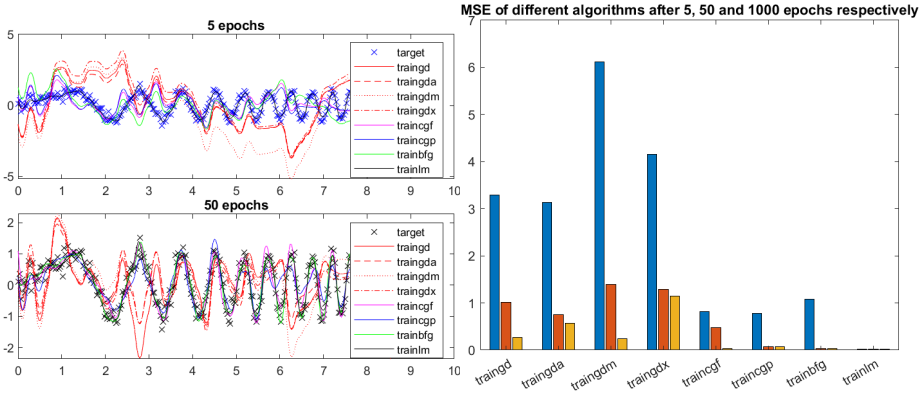

Figure 2: Comparison in fit of different algorithms with added noise.

Levenberg Marquardt has the best performance, sporting a mean squared error of

$$4.9 * 10^{-}7$$

within 10 seconds of training time using only 20 neurons.
The performance on the test set is:

$$5.7 * 10^{-}7$$

, which I think is more than sufficient and doesn't need any improvements. Figure 5 shows the test surface vs the test surface as predicted by the network.

1.2 Section 5

Without noise, Trainbr has a higher MSE than every other transfer function when the number of hidden nodes is very high(m̃ore than 200). Trainbr does however take a long time to train.
When a noise of .25 or more is added, trainlm outperforms trainbr. Trainbr took a very long time to train, oftentimes waiting more than 3 minutes to train, while trainlm trained in less than a second.

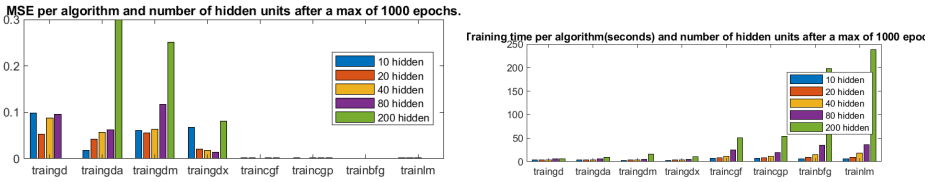

Figure 3: Comparing transfer functions and number of neurons.

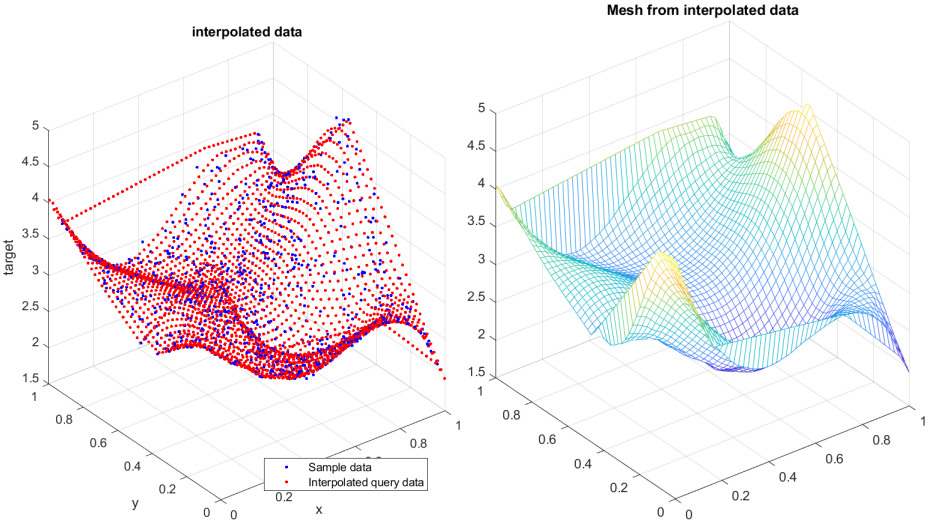

Figure 4: scatteredInterpolant interpolates the training set. The surface of the trainingset is then computed using mesh, giving a nice visualisation.

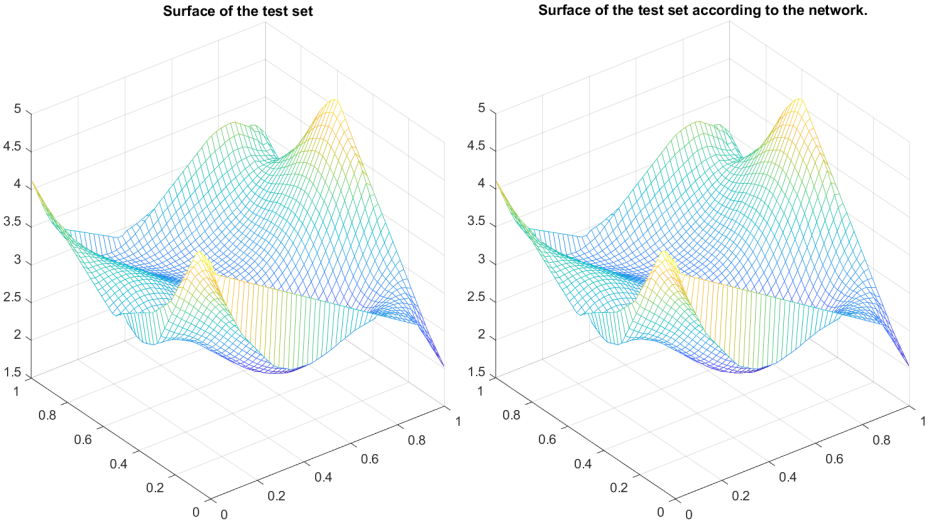

Figure 5: The test surface (left) and the test surface predicted by the network(right).

2 Session 2

2.1 hopfield network

2.1.1 Are the real attractors the same as those used to create the network? If not, why do we get these unwanted attractors? How many iterations does it typically take to reach the attractor? What can you say about the stability of the attractors?

in figure 6 there are the unwanted attractors (-1, 1), (0, 1), (0,0). There are more real attractors than used to set up the network. They surface at points of high symmetry. It typically takes less than 8 iterations to reach an attractor.

2.1.2 Do the same for a three neuron Hopfield network

Doing the same for a 3 neuron network following unwanted attractors show up: (-0.07 -1 -0.07), (0.36 -0.36 0.36) there are probably many more.

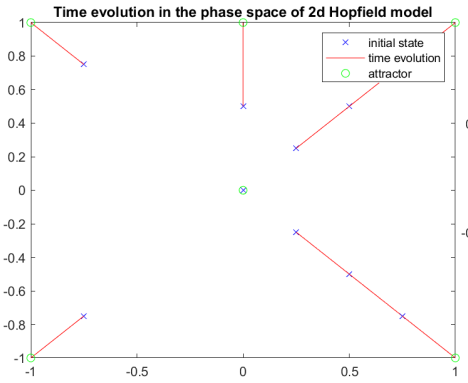

Figure 6: Unwanted attractors in the hopfield network.

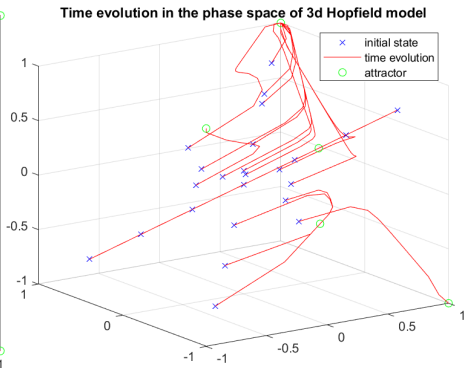

Figure 7: Unwanted attractors in the hopfield network in 3 dimensions.

2.1.3 The function hopdigit creates a Hopfield network which has as attractors the handwritten digits 0,...,9. Then to test the ability of the network to correctly retrieve these patterns some noisy digits are given to the network. Is the Hopfield model always able to reconstruct the noisy digits? If not why? What is the influence of the noise on the number of iterations?

The hopfield network can accurately recreate images that I wouldn't be able to reconstruct with my eyes anymore. the figure 8 shows the hopfield network with noise 3 and the recovered digits after 15 iterations.
When the digits are too noisy f.e. noise 8 than the hopfield network cant reconstruct them, also the higher the noise the more iterations needed to reach an attractor.

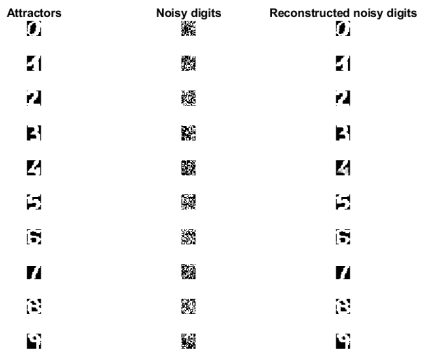

Figure 8: Hopfield network recovering digits.

2.2 Neural network

2.2.1 Train the LSTM model and explain the design process. Discuss how the model looks, the parameters that you tune, ... What is the effect of changing the lag value for the LSTM network?

Starting off, load in the santa fe dataset, 90% is allocated as training data with 10% being used to tune parameters and test our performance.
Then, for a better fit and to prevent the training from diverging, standardize the training data to have zero mean and unit variance.
To forecast the values of future time steps of a sequence, specify the responses to be the training sequences with values shifted by a x number of time steps also known as lag. The LSTM network learns to predict the value of the xth next time step. The predictors are the training sequences without the final x time step.
create an LSTM network and specify the number of hidden units. Specify the solver: adam or sgdm.

Changing the lag made my performance horrendous, when using a lag of 100 my LSTM network would just predict every target variable as 0. Changing back to a lag of 1 with 200 hidden units and the stochastic gradient ascent with momentum optimizer gave significantly better results. Resulting in an MSE of 57.4.
As can be seen in figure 10

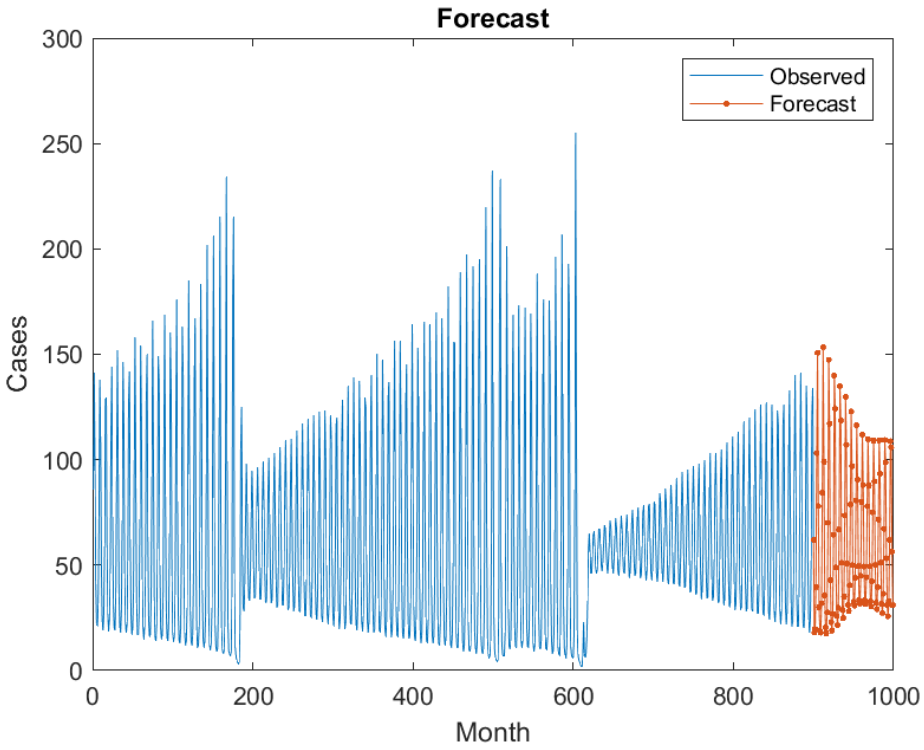

Figure 9: Forecast

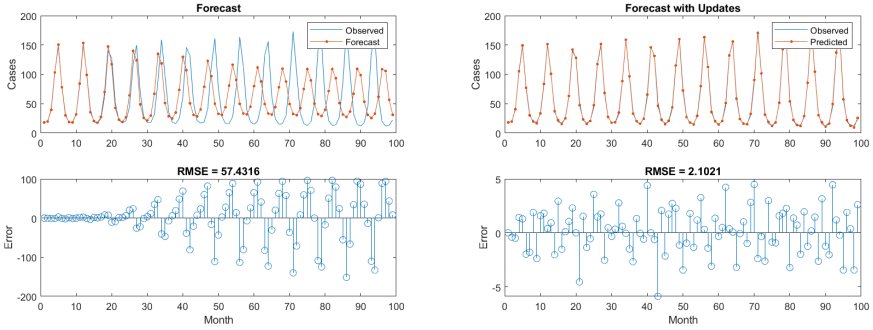

Figure 10: Forecast and RMSE left before and right after updating the network state with observed values.

3 Session 3

The 3 displayed in fig 11 is calculated by taking the mean over every pixel in the dataset.

The value of the eigenvalues drops very quickly. Looking at the images we can also see that 2 eigenvalues can recreate 3s that are clearly recognizable by a human eye, the 3,4,5th eigenvalue make the colors more intense. The plot-figure 13 of RMSE to the number of eigenvalues alligns with my observations: the picture becomes very clear from only 3 eigenvalues, with the next eigenvalues providing quickly diminishing returns. Comparing the RMSE with the cumulative sum of the eigenvalues also supports this observation. The error is proportional to the cumulative sum of the eigenvalues, the decrease in error is thus also proportional to the increase in cumulative sum.
From this exercise I conclude that we can compress correlated images to a much smaller space, using 37 eigenvalues we could store this set with a very small error(0,09). This compression can save lots of space for training data, since it keeps the features that are strongly correlated intact while easily compressing used space by 80%.
Using q=256 should give us a reconstruction of 0, but instead I got a very small error value: 6*10^-16 this is probably due to a rounding error.

3.1 Handwritten Digits PCA and reconstruction.

3.2 Digit Classification with Stacked Autoencoders and CNN

The class accuracy for the stacked network(with fine tuning) is 51% without finetuning, and 99.6% with finetuning while the accuracy for the normal neural network with 1 and 2 hidden layers is 95.6% and 96.6% respectively. A normal neural network with 3 [150,100,50] and 4 [150 150 150 100] layers gave an accuracy of 96.9% and 96.8%.
Fine tuning the stacked network allows the pre-trained network to recognize classes it was not originally trained on. the 2 hidden layers created by auto encoders might create a set of shapes and curves, but the weights are not optimized to reflect their importance in certain numbers. Backpropogation/fine tuning, will use gradient descent to give the patterns(autoencoder layers) that minimize the error for that number higher weights.

I found it impossible to get a higher accuracy with normal neural networks.

3.3 The answers to the questions in Section 3.2

Take a look at the first convolutional layer (layer 2) and at the dimension of the weights (size(convnet.Layers(2).Weights)). If you think

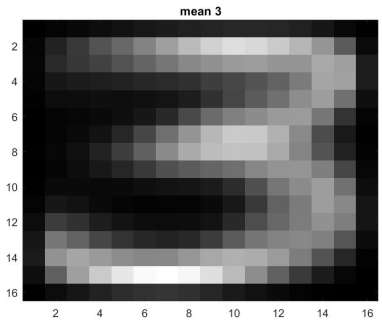

Figure 11: Mean 3 in the dataset.

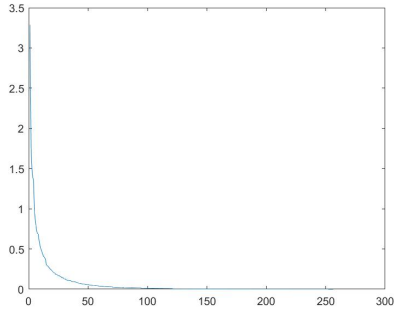

Figure 12: Value of eigenvalues ranked from large to small.

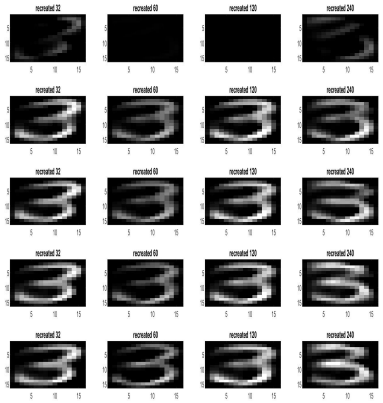

Figure 13: Comparison of image reconstruction with top 1,2,3,4,5 eigenvalues.

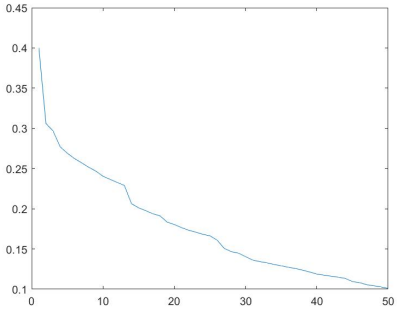

Figure 14: PCA: Comparing RMSE to the number of eigenvalues.

back at what you saw in class and/or in [5], what do these weights represent? The first Layer learned filters for capturing edge and blob features, they are primitive features that later combine in higher level image features such as corners or crosses.

Inspect layers 1 to 5. If you know that a ReLU and a Cross Channel Normalization layer do not affect the dimension of the input, what is the dimension of the input at the start of layer 6 and why? .
Layer 1 normalizes the image and has the dimensions 277x277x3
Layer 2 Convolution layer with 96 times an 11x11x3 filter with stride 4x4. (227-11)/4 + 1 = 55 new pixels, resulting in dimensions 55x55x96.
Layer 5 is a max pooling layer with filter size 3x3 and stride 2x2. The contained 9 pixels in the filter size will be converted down to 1 pixel in the output layer.

This will lead to 55-3/2 + 1 = 27 new pixels. Resulting in the dimensions 27x27x96

What is the dimension of the inputs before the final classification part of the network (i.e. before the fully connected layers)? How does this compare with the initial dimension? Briefly discuss the advantage of CNNs over fully connected networks for image classification .
The input size of the first fully connected layer, layer 17 is 9216 values. This gives us 4096x9216 weights in this layer. A fully connected network would give us the input dimension 227x227x3 = 154587 pixels. The CNN is much easier to train. Using pooling and filters, we can analyze the entire image without needing every pixel connected and reducing overfitting in the process.

The script CNNDigits.m [7] runs a small CNN on the handwritten digits dataset. Use this script to investigate some CNN architectures. Try out some different amount of layers, combinations of different kinds of layers, dimensions of the weights, etc. Briefly discuss your results. .
The accuracy starts of at 82%, doubling the amount of filters to 24 and 48 results in accuracy 97% and took 50% longer to train. Adding filters increases accuracy.
Removing the second convolutional(48 filters) and ReLu layer, whilst keeping the first convolutional layer with 24 filters gives an accuracy of 95% with the same training time as the initial model with 2 layers(12 and 24). This outperforms the initial model in every metric, some important information likely got lost in this initial model compressing to the lower dimension with 12 filters.

4 Session 4

4.1 Restricted Boltzmann machines

4.1.1 What is the effect of different parameters (# epochs and # components) on training the RBM, evaluate the performance visually by reconstructing unseen test images. Change the number of Gibbs sampling steps, can you explain the result?

.

The amount of components regulates the amount of hidden units in the model, the #epochs increases the amount of training time and by specifying how many iterations of gradient descent can be executed. Multiplying the number of components by 3 proved more powerful than multiplying the #epochs by 10. Adding more components seems to be more impactful. A low number of components-2 gives poor reconstructions of the test images no matter the number of epochs.

The influence of the # gibbs sampling steps can be seen in figure 15. In the figure of 33 gibbs samples, variations of the digits are made. Given enough steps the reconstructed digits start to look different from their original digits, mostly like 6s and 9s.

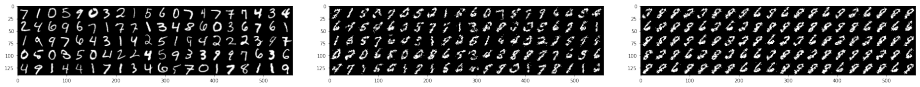

Figure 15: #gibbs sampling steps and effect on reconstruction of unseen test images. 0, 33 and 1000 sampling steps respectively.

4.1.2 Use the RBM to reconstruct missing parts of images. What is the role of the number of hidden units, learning rate and number of iterations on the performance. What is the effect of removing more rows on the ability of the network to reconstruct? What if you remove rows on different locations (top, middle,...)?

.

Increasing the number of hidden units from 10 to 33 to 60 reduces the noise in the reconstructed image. Increasing the number of epochs from 30 to 200 provides only minor improvements.
Removing more rows results in reduced reconstruction quality, the RDM struggles with recreating the top parts of the 7 and the 2 and has an easier time reconstructing the middle part of a 1 or 7.

4.2 Deep Boltzmann Machines

4.2.1 Load the pretrained DBM that is trained on the MNIST database. Show the filters (interconnection weights) extracted from the previously trained RBM (see exercise 1) and the DBM, what is the difference? Can you explain the difference between filters of the first and second layer of the DBM?

.
The filters of the RBM (60 components) and DBM are shown in figure 16. The RBM looks to be filtering for for lines and points, the DBM has similar behavior in the first layer. The second layer of the DBM considers structures in the entire image.
The structures in the 2nd layer come from a combination of lines and points from the first layer.

4.2.2 Sample new images from the DBM. Is the quality better than the RBM from the previous exercise and explain why?

The DBM reconstructs the images significantly better, comparing figure 17 to 15, the DBM reconstructs recognizable digits, while the RBM can not. The increased complexity in the model of the DBM is to thank for the increase in quality.

4.3 Generative Adversarial Networks

4.3.1 Select one class from the CIFAR dataset and train a Deep convolutional generative adversarial network (DCGAN). Take into account the architecture guidelines from Radford et al. [5]. Make sure that you train the model long enough, such that it is able to generate "real" images. Monitor the loss and accuracy of the generator vs discriminator, and comment on the stability of the training. Explain this in context of the GAN framework.

A GAN has two elements working together: the generator and the discriminator. The generator creates fake pictures of objects and the discriminator tries to guess if the image is real or fake.
I trained the GAN on class 1 (cars) with batches of size 200 over 40000 steps. The loss functions and accuracy are shown in figure 18.
I would've wanted to make the graphs more clear by splitting them on accuracy and loss, but my runtime disconnected and I ran out of GPU time.
What I can still deduce from this graph is the following. The stability of training is unstable cause we're training two models simultaneously that compete against each other. During the execution of the algorithm the generator's accuracy increases while the discriminators accuracy drops. After enough iterations, training has an adverse effect on the performance of the generator, at batch

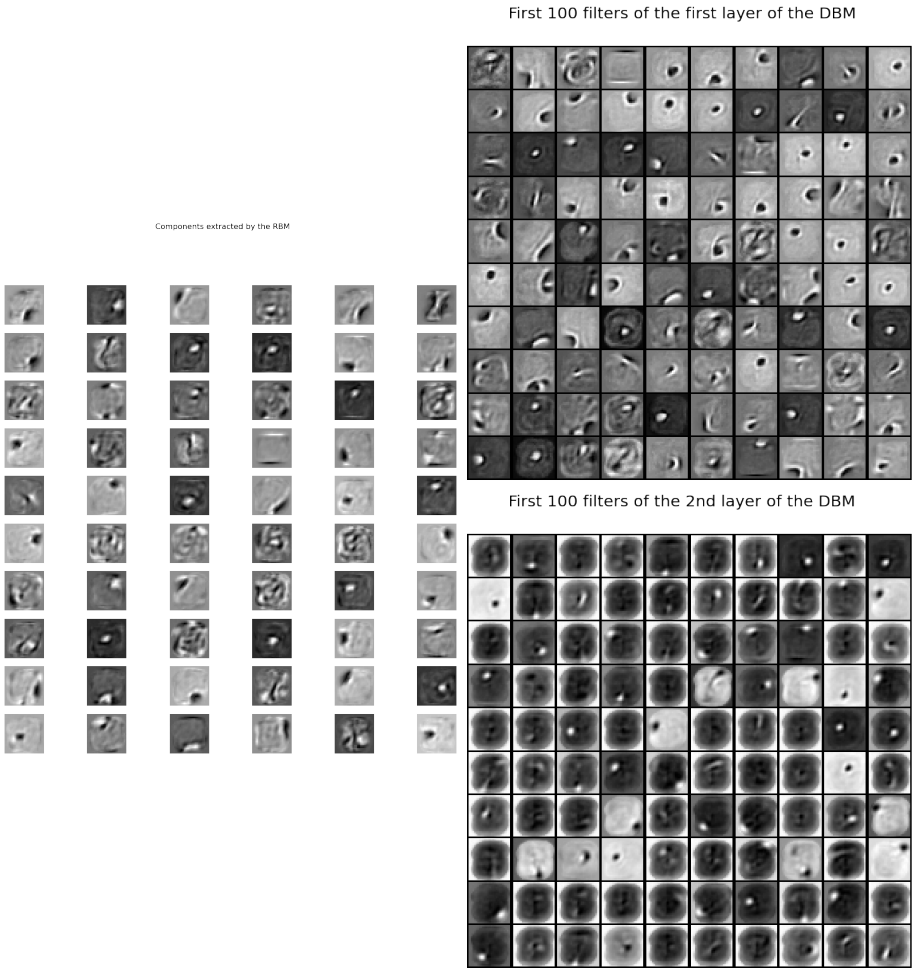

Figure 16: filters of RBM vs DBM

25000 the discriminator begins to guess quasi random since the images start to look very much alike. With the generator trying to adapt to the random feedback of the discriminator, its quality start to collapse.

Figure 17: Reconstruction after 100 gibbs sampling steps by DBM.

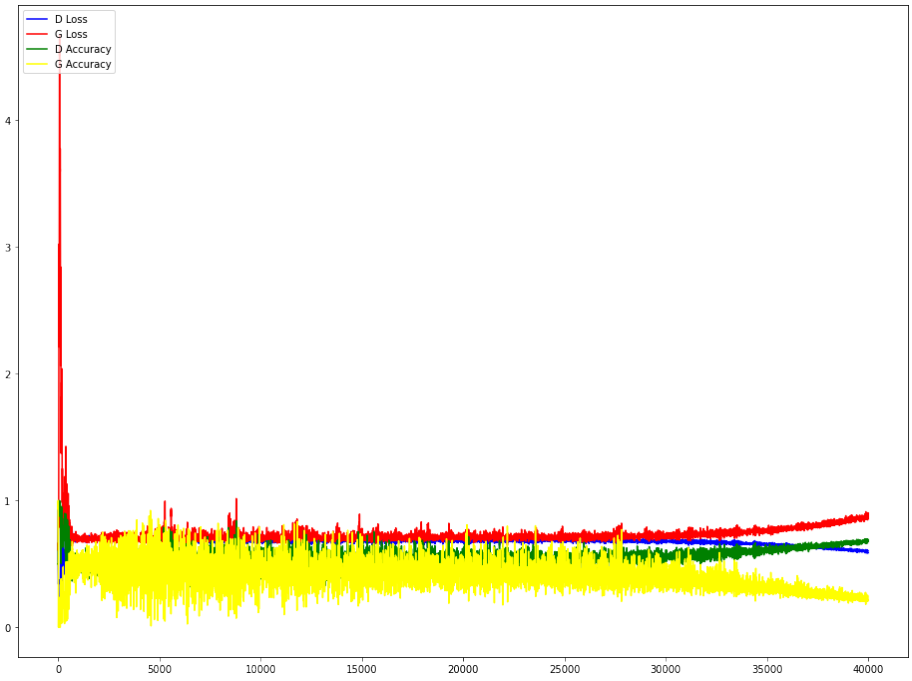

Figure 18: Loss and accuracy of the GAN.

4.4 Optimal transport

4.4.1 Upload your own images (of equal size) using the Files tab. Afterwards transfer the colors between the two images using the provided notebook. Show the results and explain how the color histograms are transported, how is this different from non optimal color swapping (e.g. just swapping the pixels)?

By transferring the histograms you transfer the color palette to another image while keeping the objects. Earth movers distance transports the histogram 'pile

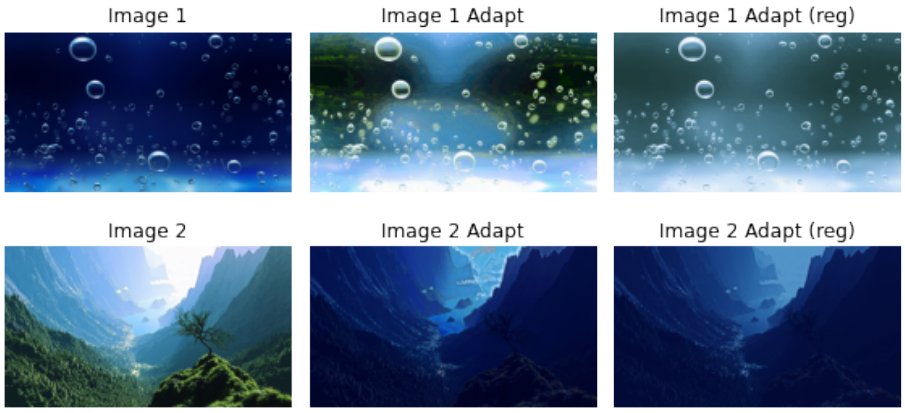

Figure 19: Transfered colors of two images, Adapt uses EMD and Adapt (reg) uses sinkhoorn.

of sand' from one image to the other with the least amount of work. As you can see in figure 19 it transports the green from image 2 to 1 over too intense, making it unnatural. The regularisation from sinkhoorn reduces this intensity, making the image more realistic but reducing the intensity of the colors.
Optimal transport results in higher image quality, with pixel swapping there would be more artifacts. The exact colors are swapped so the bubble picture might suddenly get a mixture of green and light blue even though the colors were full blue in the original. It also creates a smoother image, since with pixel swapping we would see some pixelated borders.

4.4.2 Train a fully connected minimax GAN and Wasserstein GAN on the MNIST dataset. Compare the performance of the two GAN's over the different iterations. Do you see an improvement in stability and quality of the generated samples? Elaborate on the knowledge you have gained about optimal transport and the Wasserstein distance.

I trained the minimax GAN and wasserstein GAN with weight clipping. Looking at figure 21 and 23, the minimax GAN has some background noise and thats still there even in the final iteration, while the wasserstein GAN has some uncompleted shapes (digit 5 and 6 in particular).
In the final iteration there seems to be some collapse of the model cause there is little diversity.
Wasserstein and minimax have a different loss function, which is clearly visible in the loss graphs. The wasserstein GAN loss function is an approximation of the wasserstein metric/earth movers distance, where the minimax GAN uses a sigmoid to become probabilities.

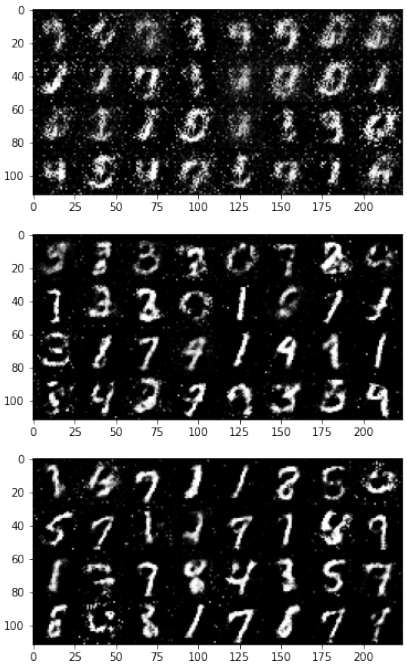

Figure 20: early, middle, final iteration minimax GAN

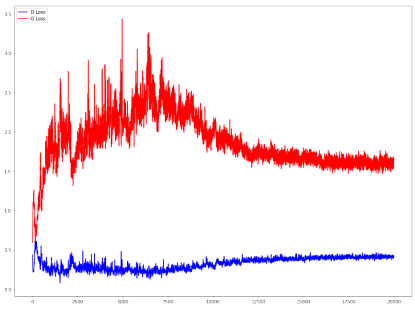

Figure 21: Loss minimax GAN

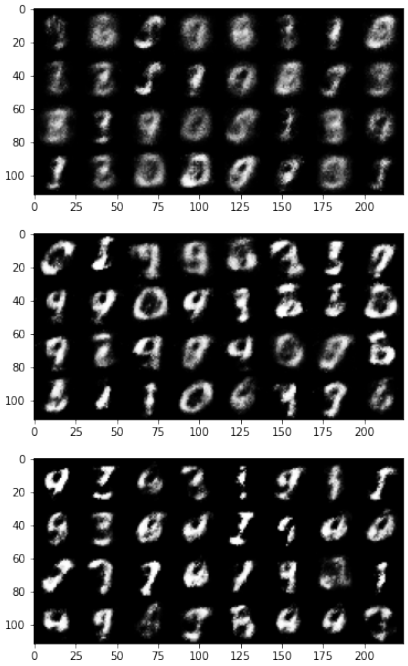

Figure 22: early, middle, final iteration wasserstein GAN

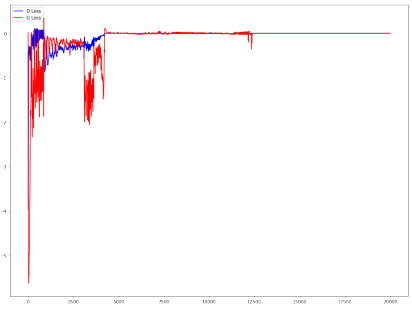

Figure 23: Loss Wasserstein GAN