

Artificial Neural Networks: Exercise session 4

Stijn Staring (r0620003)

May 23, 2021

1 Restricted Boltzmann Machines

The Boltzmann machine is a parametrized generative model representing a probability distribution. It is a neural network that belongs to energy based models. A Boltzmann Machine consists out of a layer of visible units v and hidden units h . The energy function used by the model is shown in Eq. 1a and is part of the joint probability distribution. The standard type of Boltzmann Machines, uses binary-valued hidden and visible units and is therefore convenient to be used for black white images. The difference between a Restricted Boltzmann Machine and a conventional Boltzmann Machine, is that the units of one layer are not connected with each other. Because of this restriction, the partition function in the joint probability distribution seen in Eq. 1b becomes tractable. A RBM trains in an unsupervised manner using only the observations and calculating the gradient of the log-likelihood of the observations with respect to the weight matrix W , and bias terms a and b . Training a RBM means adjusting the RBM weight values and biases such that the probability distribution $P(v)$ fits the training data as well as possible. The gradient of the log-likelihood is calculated by the difference between the data expectation and model expectation and is used together with a learning rate and a gradient decent update strategy. However, because the calculation of the model expectation is intractable, the Contrastive Divergence algorithm is used. Gibbs sampling is applied to generate an observation from the model distribution and is used in the Contrastive Divergence algorithm. Gibbs sampling is initialized by a known observation

$$E(v, h) = -v^T W h - b^T v - a^T h, \quad (1a)$$

$$P(v, h) = \frac{1}{Z} e^{-E(v, h)}. \quad (1b)$$

1.1 Hyperparameter tuning

A small hyperparameter tuning is displayed in Table 1. It is seen that more iterations give better results. When looking at the column with 20 iterations, an U-shape is noticed in the pseudo-likelihood. For a small learning rate of 0.005 a larger error is found due to the slow convergence and a larger learning rate of 0.1 performs worse due to oscillations during the application of gradient descent. The use of more hidden units gives an higher likelihood of the observations. Using more iterations and hidden units increases the calculation load of the model. The final chosen values for the parameters are 20 iterations, 20 hidden units and a learning rate of 0.01. In Figure 2 the comparison on the reconstruction of the number 4 is given with the tuned parameters and the default ones. It shows that the tuned RBM is able to better reconstruct an image with deleted row than when using the default parameter values that uses 10 iterations, 10 hidden units and a learning rate of 0.01 during training.

Learning rate/Iterations	5	10	20	Iterations/Hidden units	5	10	20
0,005	-141	-134	-127,41	5	-185	-155,08	-134,46
0,01	-166	-147,66	-125,55	10	-186	-154	-130
0,05	-141,15	-134,35	-127,41				
0,1	-134,46	-130,24	-130,81				

Table 1: Pseudo-likelihood on the training data when training with different parameter values.

1.2 Gibbs sampling

In Gibbs sampling it is started from an initial known observation $v^{(1)}$ and by the sequential use of $P(h|v^{(1)})$ and $P(v|h^{(1)})$, one iteration of the Gibbs sampling is completed and a new observation is obtained. A property of Gibbs sampling is that only after a infinite amount of iterations it can be guaranteed that the generated sample

originates from the model probability distribution that is tried to be sampled from. As can be seen in Figure 1 after one iteration the original number 2 is still visible but after 100 iterations it is not recognizable anymore. The influence of the initial observation is reduced after 100 iterations and the obtained observation will correspond better to one that originates from the model probability distribution.

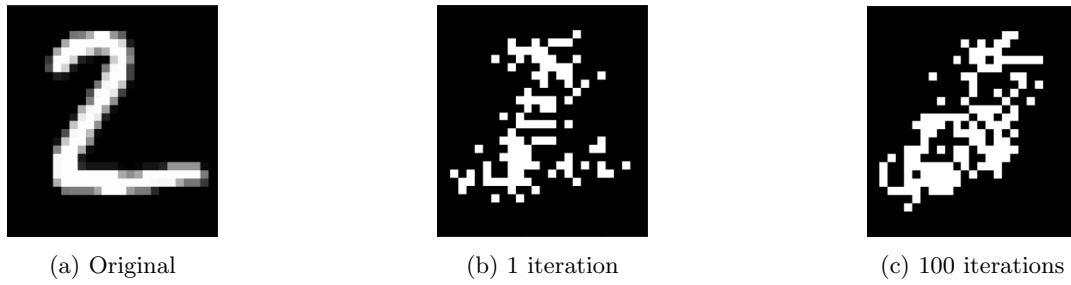


Figure 1: The effect of iterations of Gibbs sampling.

1.3 Reconstructing images

The general rule in reconstructing images is that when more information is missing from the original image, it is harder to reconstruct. Deleting more information can be done by deleting more rows or deleting crucial places that play a key role in distinguishing the number. Figure 2 shows the reconstruction of the number 4 using the default RBM and tuned RBM when rows 10 till 15 are removed. The tuned model shows better results. In order to reconstruct the image, the image with deleted rows is feeded as observation to the trained RBM and Gibbs sampling is applied. An observation is retrieved from the model probability distribution and the corresponding rows are used to fill the gap of the original image. For both images 10 Gibbs sampling steps are used.

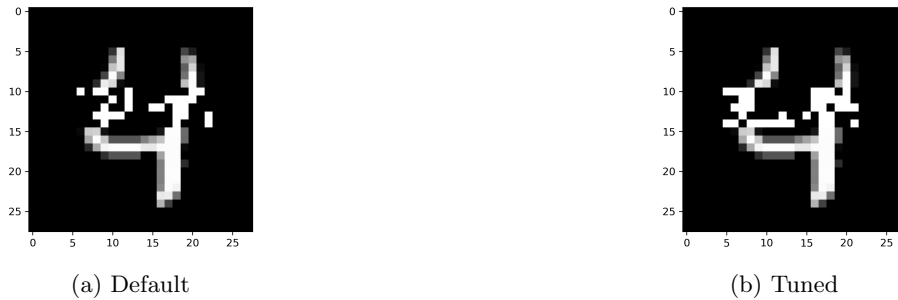


Figure 2: The effect of hyper parameter tuning.

2 Deep Boltzmann Machines

A Deep Boltzmann Machines is still restricted, which means that the nodes of the network of one layer are not connected. However, the difference with a shallow RBM is that it has multiple layers of hidden units stacked on top of each other. Due to this, a DBM can learn features that are non-linear combinations of features extracted in the lower layers. In comparison, the RBM can only extract features directly from the pixels. Therefore, more advanced features can be learned by a DBM. Figure 3 shows the weights of the hidden units of the RBM and two layers of the DBM. It is clear that both RBM and DBM show similar extracted features in layer 1. In the second layer more complex features are found that the RBM hasn't captured.

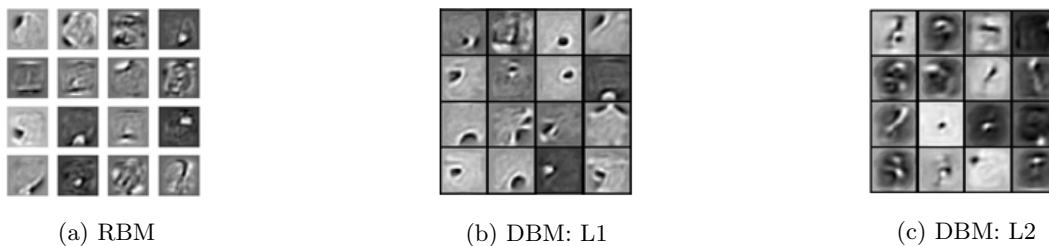
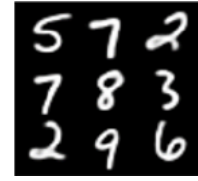


Figure 3: Weights of the units of a RBM and DBM.

That the DBM can better capture the important features of the digit is shown in Figure 5 where it is seen that the images originating from a DBM are much more clear. For the generation of the images 1 Gibbs sampling iteration is used.



(a) RBM



(b) DBM

Figure 4: Generating digits with a RBM and DBM.

3 Generative Adversarial Networks

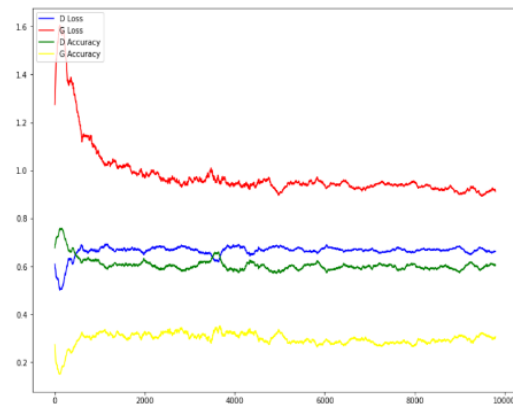
A Generative adversarial network is an unsupervised learning task where regularities and patterns are learned in the input data. Afterwards the model can be used to generate examples that could have been part of the original training set. It is important that the generated examples are not exactly the same as the training examples because these are already available. A GAN should learn the underlying probability distribution of the training examples and sample from it to generate new examples. A GAN is trained making use of two adversarial neural networks that play a zero sum game against each other. Player one is the generative model that generates fake training examples from noise and player two is a discriminator that assesses if the presented example is a fake or real training example by assigning probabilities. When the discriminator is fooled around half of the times, training is stopped because this means that the generator generates plausible examples. By making use of the interplay between the two players, the learning problem is framed in a supervised learning problem. Neural networks used inside a GAN as generator is for example a CNN. Here no pooling is used because dimensionality reduction is not desired. Also no fully connected layers are used. The disadvantage of using GAN is that it can become unstable when learning in practise. Applications of GAN are:

- Explainable AI
- Augmentation of a dataset

When a model can generate logical data, the user puts more trust in it and it contributes to the explainability of the model. Also, GANs can be used in a context where there is only a small amount of data available and where it can be augmented with generated examples. A very famous example is the use of GAN in deepfakes where it is possible to generate a face on the movements of a person so that it looks like somebody else is doing or saying something in a video.



(a) Generated bird images



(b) Loss and accuracy of the generator vs discriminator

Figure 5: Generating digits with a RBM and DBM.

Figure 5a shows the generated examples of images of birds wherefore a GAN was trained on the CIFAR dataset. Figure 5b shows the accuracy and loss of the generator in respectively yellow and red. The accuracy and loss of the discriminator are shown in respectively green and blue. It is observed that the four curves are correlated. This is logically because the Generator and Discriminator play a game against each other. When the Generator loss becomes higher, the Discriminator loss becomes lower. This can be clearly seen in the opposite direction of the red and blue curves. The loss functions behave opposite to the accuracy and therefore yellow and red, and green and blue also show opposite peaks. It can be seen that the accuracy and loss functions all converge which means that the model didn't become unstable. During training the effect of "mode collapse" could occur which

means that the generator has found one image to always successfully mislead the discriminator with. Therefore, the generator starts converging to a situation where it always uses this particular image. The accuracy of the generator will become zero.

4 Optimal transport

Optimal transport wants to change one probability distribution into another, while making minimal changes to the original one. To calculate distance between the two distributions, the Wasserstein distance is used. The Wasserstein distance measures the distance between two probability distributions by calculating the minimum of changes that have to be made to turn the first one into the second. During training of the optimal transport problem, this distance is tried to be minimized. If discrete distributions are considered, the Wasserstein distance corresponds to the minimum of a linear program. A variant on the Wasserstein distance is the Sinkhorn distance where the amount of information entropy is added as a negative cost in the objective function of the distance metric and which serves as regulation term.

4.1 Colour swapping

Colour swapping of images is performed by creating a start set and a target set by randomly sampling respectively both the original pictures. Each sample corresponds to a random pixel consisting out of three float numbers that correspond to a colour according to the RGB colour code. Because random sampling is performed all the samples in the start set and target set belong to a different probability distribution. The task during optimal transport is to learn how to transform the start distribution to the target distribution. The function used to evaluate the learned relation between the two probability distributions is the Wasserstein and Sinkhorn distance. Results of the colour swapping are displayed in Figure 6. It can be seen that the darkness of the colours are preserved causing the contours to be very visible. Using the Sinkhorn distance leads to a more homogenous colour distribution, which is in accordance to the highest amount of information entropy. When a non-optimal method is used, original image won't be recognizable.

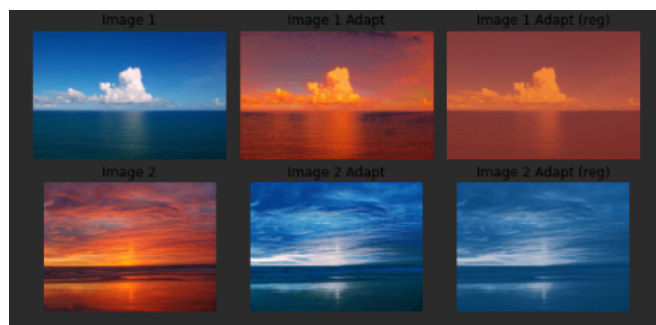


Figure 6: Swapping colour using OT. Pictures in order are: original, OT Wasserstein, OT Sinkhorn

4.2 WGAN

This section shows the different generated digits making use of a GAN with a generator and discriminator with fully connected layers. A standard GAN making use of log-sigmoid functions as cost functions for the distance between the probability distribution of the data and the model. The WGAN omits this and uses the Wasserstein distance metric instead. In a Wasserstein GAN, the discriminator emits an unconstrained real number instead of a probability and performs the role of a critic. Two variants of WGAN are tried. WGAN with weight clipping and gradient penalization. From Figure 7 it is clear that using WGAN gives smoother and less noisy generated digits. No mode collapse occurs.

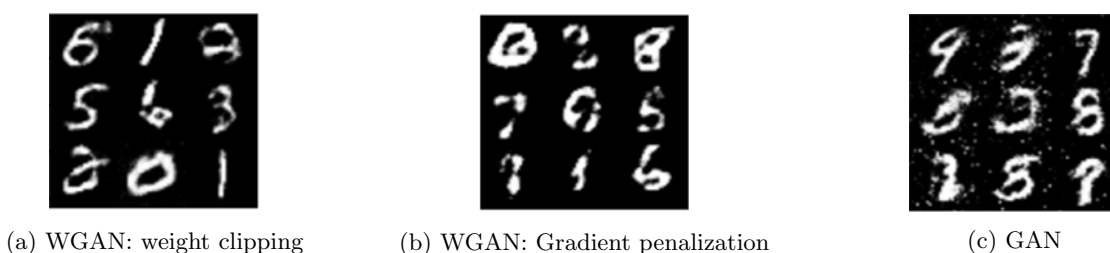


Figure 7: Generated images from WGAN and GAN using 20000 batches.