Artificial neural networks - Exercise session 4

Generative models

Bram De Cooman, Hannes De Meulemeester, Joachim Schreurs and David Winant

2020-2021

1 Restriced Boltzmann Machines

1.1 Introduction

Restricted Boltzmann machines (RBMs) are probabilistic graphical models that can be interpreted as stochastic neural networks [1, 2]. A RBM is a parameterized generative model representing a probability distribution. Given some observations, the training data, learning a RBM means adjusting the RBM parameters such that the probability distribution represented by the RBM fits the training data as well as possible. The standard type of RBM has binary-valued hidden and visible units, and consists of a matrix of weights $W = (w_{i,j}) \in \mathbb{R}^{m\times n}$ associated with the connection between hidden unit h_j and visible unit v_i, as well as bias weights (offsets) a_i for the visible units and b_j for the hidden units. Given these, the energy of a configuration is defined as:

$$E(v,h) = -\sum_{i,j} v_i w_{i,j} h_j - \sum_i a_i v_i - \sum_j b_j h_j.$$

This energy function is analogous to that of a Hopfield network. As in general Boltzmann machines, probability distributions over hidden and/or visible vectors are defined in terms of the energy function. Where the probability function is equal to:

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

where Z is a partition function defined as the sum of $e^{-E(v,h)}$ over all possible configurations (in other words, just a normalizing constant to ensure the probability distribution sums to 1). Similarly, the marginal probability of a visible (input) vector of booleans is the sum over all possible hidden layer configurations:

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

The visible and hidden neurons can be thought of as being arranged in two layers. The visible units constitute the first layer and correspond to the components of an observation (e.g., one visible unit for each pixel of a digital input image). The hidden units model dependencies between the components of observations (e.g., dependencies between pixels in images). They can be viewed as non-linear feature detectors. A schematic representation is visible on Figure 1.

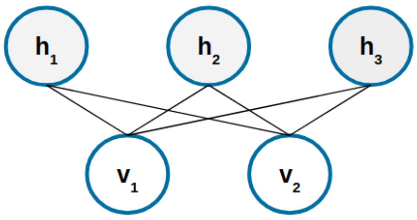

Figure 1: Schematic representation of the Restricted Boltzmann machine.

After successful learning, an RBM provides a closed-form representation of the distribution underlying the observations. It can be used to compare the probabilities of (unseen) observations and to sample from the learned distribution (e.g., to generate new images), in particular from marginal distributions of interest. For example, we can fix some visible units corresponding to a partial observation and sample the remaining visible units for completing the observation (e.g., to solve an image inpainting task). This is done by using the conditional probability. That is, for m visible units and n hidden units, the conditional probability of a configuration of the visible units v, given a configuration of the hidden units h, is:

$$P(v|h) = \prod_{i=1}^{m} P(v_i|h).$$

Conversely, the conditional probability of h given v is given by:

$$P(h|v) = \prod_{j=1}^{n} P(h_j|v).$$

The individual activation probabilities are given by:

$$\begin{aligned} P(h_j = 1|v) &= \sigma\left(b_j + \sum_{i=1}^{m} w_{i,j} v_i\right) \\ P(v_i = 1|h) &= \sigma\left(a_i + \sum_{j=1}^{n} w_{i,j} h_j\right), \end{aligned}$$

where σ denotes the logistic sigmoid. The term **restricted** comes from the fact that there no hidden-to-hidden and visible-to-visible connections. If we also allow these, we get a Boltzmann machine.

1.2 Exercises

The exercises consist of python notebooks that you will run in Google Colab (`https://colab.research.google.com`/). Upload the file `RBM.ipynb` using the `"upload tab"`. Navigate trough the different cells with `Run` icon or pressing `Ctrl + enter`. A more elaborate introduction can be found at `https://colab.research.google.com/notebooks/intro.ipynb`. Afterwards, answer the following questions:

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?
2. Use the RBM to reconstruct missing parts of images[1]. 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,...)?

[1]An interesting video about image reconstruction using RBM's can be found at `https://www.youtube.com/watch?v=tk9FTdKOL5Q`

2 Deep Boltzmann Machines

2.1 Introduction

Deep Boltzmann machines can be understood as a series of restricted Boltzmann machines stacked on top of each other [3]. The hidden units are grouped into a hierarchy of layers, such that there is full connectivity between subsequent layers, but no connectivity within layers or between non-neighbouring layers. A schematic representation is visible on Figure 2.

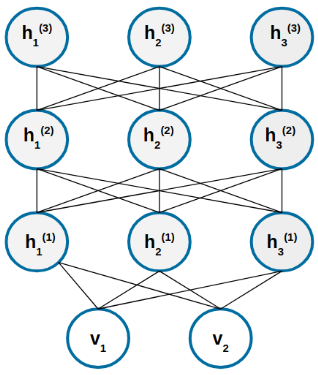

Figure 2: Schematic representation of a 3 layer deep Boltzmann machine.

2.2 Exercises

Upload the file `DBM.ipynb` and go trough the code. Afterwards, answer the following questions:

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

2.3 Optional: High Quality Image Generation with StyleGAN

The final section of the `DCGAN.ipynb` notebook illustrates state-of-the-art image generation using a StyleGAN with differentiable augmentations [8, 9]. The provided script allows you to experiment with training a StyleGAN on your own images. The code will output intermediate samples so you can observe the training process.

3 Generative Adversarial Networks

3.1 Introduction

Generative adversarial networks (GANs) are a class of algorithms used in unsupervised machine learning, implemented by a system of two neural networks competing with each other in a zero-sum game framework [4]. One neural network, called the generator, generates new data instances, while the other, the discriminator, evaluates them for authenticity; i.e. the discriminator decides whether each instance of data belongs to the actual training dataset or not.

To summarize, here are the steps a GAN takes for an image generation example:

1. The generator takes in random numbers and returns an image.

2. This generated image is fed into the discriminator together with a batch of images taken from the actual dataset.

3. The discriminator takes in both real and fake images and returns probabilities, a number between 0 and 1, with 1 representing a prediction of authenticity and 0 representing fake.

4. Update the weights of the competing neural networks.

3.2 Exercises

Upload the file `DCGAN.ipynb` and go trough the code. Afterwards, answer the following questions:

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.

4 Optimal transport

Optimal transport (OT) [6] theory can be informally described using the words of Gaspard Monge (1746-1818): A worker with a shovel in hand has to move a large pile of sand lying on a construction site. The goal of the worker is to construct with all that sand a target pile with a prescribed shape (for example, that of a giant sand castle). Naturally, the worker wishes to minimize her total effort, quantified for instance as the total distance or time spent carrying shovels of sand. People interested in OT cast that problem as that of comparing two probability distributions-two different piles of sand of the same volume. They consider all of the many possible ways to morph, transport or reshape the first pile into the second, and associate a ”global” cost to every such transport, using the ”local” consideration of how much it costs to move a grain of sand from one place to another. In OT, one analyzes the properties of that least costly transport, as well as its efficient computation. An example of the computation of OT and displacement interpolation between two 1-D measures is visible on Figure 3.

A common problem that is solved by OT is the assignment problem. Suppose that we have a collection of n factories, and a collection of n stores which use the goods that the factory produce. Suppose that we have a cost function c, so that $c(x, y)$ is the cost of transporting one shipment of the factory from x to y. For simplicity, we ignore the time taken to do the transporting and a factory can only deliver complete goods (no splitting of goods). Let us introduce some notation so we can formally state this as an optimization problem. Let r be the vector containing the amount of goods every store needs. Similarly, k denotes the vector of how much goods every factory produces. Often r and k represent marginal probability distributions, hence their values sum to one. We wish to find the optimal transport plan, whose total cost is equal to:

$$d_M(r,k) = \min_{P\in U(r,k)} \sum_{ij} P_{ij} M_{ij}, \tag{1}$$

where M is the cost matrix, $U(r, k)$ all possible ways to match factories with stores and P_{ij} quantifies the amount of goods that is transported from factory i to store j. This is called the optimal transport between r and k. It can be solved relatively easily using linear programming. The optimum, $d_M(r, k)$, is called the Wasserstein metric. It is a distance between two probability distributions, sometimes also called the earth mover distance as it can be interpreted as how much ’dirt’ you have to move to change one ’landscape’ (distribution) in another (see Monge’s original problem).

Consider a slightly modified form of optimal transport:

$$d_M^\lambda(r,k) = \min_{P\in U(r,k)} \sum_{ij} P_{ij} M_{ij} - \frac{1}{\lambda} h(P), \text{with}\, h(P) = -\sum_{ij} P_{ij} \log(P_{ij}). \tag{2}$$

Which is called the Sinkhorn distance, where the second term denotes the information entropy of P. One can increase the entropy by making the distribution more homogeneous, i.e. giving everybody a more equal share of goods. The parameter λ determines

[1]Above explanation and figures are based on [6].

the trade-off between the two terms: trying to give every store only goods from the closest factory (lowest value in the cost matrix) or encouraging equal distributions. This is similar to regularization in, for example, ridge regression. Similar as that for machine learning problems a tiny bit of shrinkage of the parameter can lead to an improved performance, the Sinkhorn distance is also observed to work better than the Wasserstein distance on some problems. This is because we use a very natural prior on the distribution matrix P: in absence of a cost, everything should be homogeneous.

In many situations the primary interest is not to obtain the optimal transportation map. Instead, we are often interested in using the optimal transportation cost as a statistical divergence between two probability distributions. A statistical divergence is a function that takes two probability distributions as input and outputs a non-negative number that is zero if and only if the two distributions are identical. Statistical divergences such as the KL divergence are frequently used in statistics and machine learning as a way of measuring dissimilarity between two probability distributions. For example, suppose you want to compare different recipes, where every recipe is a set of different ingredients. There is a meaningful distance or similarity between two ingredients, but how do you compare the recipes themselves? Using optimal transport boils down to finding the effort needed to turn one recipe into another.

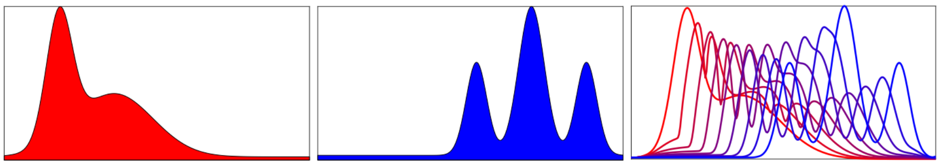

Figure 3: Example of the computation of OT between two 1-D measures. The third figure shows the displacement interpolation between the two using OT.

4.1 Exercises

Upload the file `OT.ipynb` and go trough the code[2]. Afterwards, answer the following question:

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)?

Upload the file `WGAN.ipynb` and go trough the code. Afterwards, try to answer the following question:

1. 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.

5 Report

Write a report of maximum 4 pages (including text and figures) to discuss the exercises in sections 1,2,3 and 4.

References

[1] Fischer, A. and Igel, C. (2012, September). An introduction to restricted Boltzmann machines. In Iberoamerican Congress on Pattern Recognition (pp. 14-36). Springer, Berlin, Heidelberg.

[2] Hinton, G. E. (2012). A practical guide to training restricted Boltzmann machines. In Neural networks: Tricks of the trade (pp. 599-619). Springer, Berlin, Heidelberg.

[2]The code is taken from the OT toolbox `https://pot.readthedocs.io/en/stable/`.

[3] Salakhutdinov, R. and Larochelle, H. (2010, March). Efficient learning of deep Boltzmann machines. In Proceedings of the thirteenth international conference on artificial intelligence and statistics (pp. 693-700).

[4] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S. and Bengio, Y. (2014). Generative adversarial nets. In Advances in neural information processing systems (pp. 2672-2680).

[5] Radford, Alec, Luke Metz, and Soumith Chintala. ”Unsupervised representation learning with deep convolutional generative adversarial networks.” arXiv preprint arXiv:1511.06434 (2015).

[6] Peyré, Gabriel, and Marco Cuturi. ”Computational optimal transport.” Foundations and Trends® in Machine Learning 11.5-6 (2019): 355-607.

[7] Brock, Andrew, Jeff Donahue, and Karen Simonyan. ”Large scale GAN training for high fidelity natural image synthesis.” arXiv preprint arXiv:1809.11096 (2018).

[8] Karras, Tero, Samuli Laine, and Timo Aila. ”A style-based generator architecture for generative adversarial networks.” Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2019.

[9] Zhao, Shengyu, et al. ”Differentiable Augmentation for Data-Efficient GAN Training.” Advances in Neural Information Processing Systems 33 (2020).