Generative Models

James O'Reilly
james.oreilly@student.kuleuven.be

Restricted Boltzmann Machines

Introduction to RBMs

A detailed introduction to restricted Boltzmann machines (RBMs) is given in Fischer and Igel.[1] In short, an RBM is a generative stochastic artificial neural network that can learn a probability distribution over its set of inputs. 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. Below we investigate the effects of changing different training parameters when training RBMs (number of epochs, number of components, Gibbs sampling steps, etc). A practical guide for training RBMs is given by Hinton.[3]

Exercises

In this section, we investigate the effect of changing training parameters for an RBM that is trained on the MNIST dataset. Increasing the number of iterations (number of forward and backward passes through the network) or increasing the number of components (hidden units) should increase the performance of the network. Note that RBMs are particularly difficult to train. As the partition function which normalises the probability function is intractable, we cannot estimate the log-likelihood $\log(P(x))$ during training, and therefore have no direct metric for choosing network hyper-parameters. Instead, more tractable functions can be used as a proxy to the actual likelihood. One example is the pseudo-likelihood, which will be used here.

As the number of iterations was increased, there was a steady increase in the pseudo-likelihood score. This increase eventually plateaued after certain number of iterations. Figure 1 shows the pseudo-likelihood over 30 iterations, measured for a network with 20 hidden units and a learning rate of 0.01. The pseudo-likelihood also increases as the number of hidden units increases.

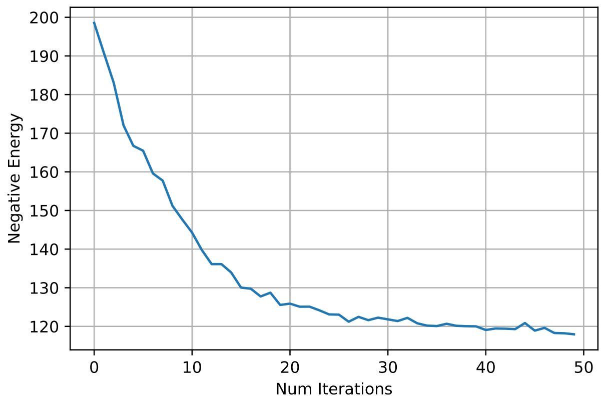


Figure 1: Negative energy vs iterations. Here the negative of the pseudo-likelihood score is used for clarity of presentation. Note the plateau after a certain number of iterations.

Samples are obtained from the test image using Gibbs sampling. The results for different numbers of Gibbs

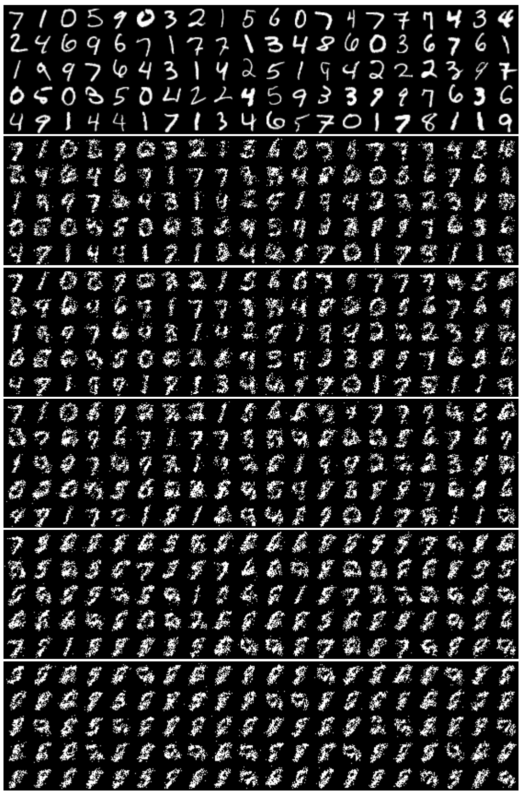


Figure 2: Results for different Gibbs sample steps. The first figure is the original test image. The images below are for steps of 1, 10, 100, 1000, and 10000, respectively.

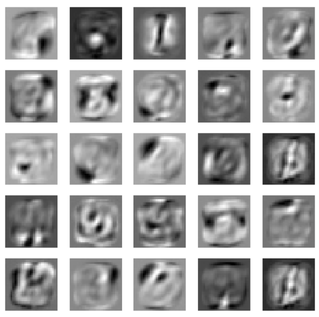
sample steps are displayed below, and compared with the original test data. Looking at 2, it can be seen that as the number of sample steps increases, the figures become less clear and more homogeneous. I'm not sure why this is the case.

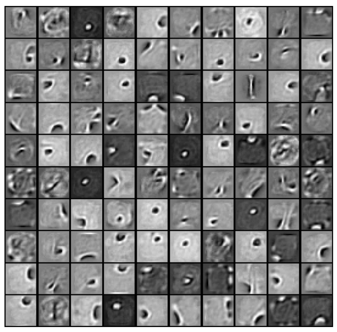
Reconstruction of Unseen Images

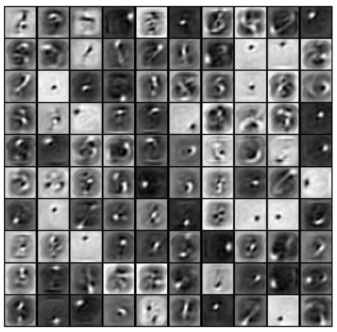
Here we remove pixel rows from certain images. Using the recently trained RBM, we will then try to reconstruct the original images. By changing the number and position of the rows that are removed, we can observe differences in the ability of the RBM to reconstruct the original images. Increasing both the number of hidden units and the number of iterations improved the ability of the network to reproduce the original images. Even with very few rows removed, reconstruction with the shallow RBM with 20 hidden units trained for 30 epochs was rather poor. Removing more than seven rows in any position resulted in poor reconstruction. Reconstruction was improved if the rows were removed from the center of the image, perhaps because the defining characteristics some digits are located at the top and bottom of the image.

Deep Boltzmann Machines

Deep Boltzmann machines can be understood as a series of restricted Boltzmann machines stacked on top of each other.[5] 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 DBM that was pre-trained on the MNIST database was used in this assignment. The interconnection weights from the pre-trained DBM can be extracted and compared with those from the RBM we trained previously. In the first layer of the DBM, the filters focus heavily on identifying loops and teh position of these loops. The area within these loops is black. While in the RBM, the filters contain larger, more complex structures with loops, lines, and differences in brightness throughout the image. The RBM filters are aslo much more noisy. This is the case because the DBM has multiple layers, and so the earlier layers can focus on identifying smaller, mores specific features which can then be combined in the later layers. This is not possible in the RBM, as it only has one layer, and so it must attempt to capture lots of different features (lines, loops, curves, etc) in the this single


(a) RBM hidden layer


(b) DBM layer one


(c) DBM layer 2

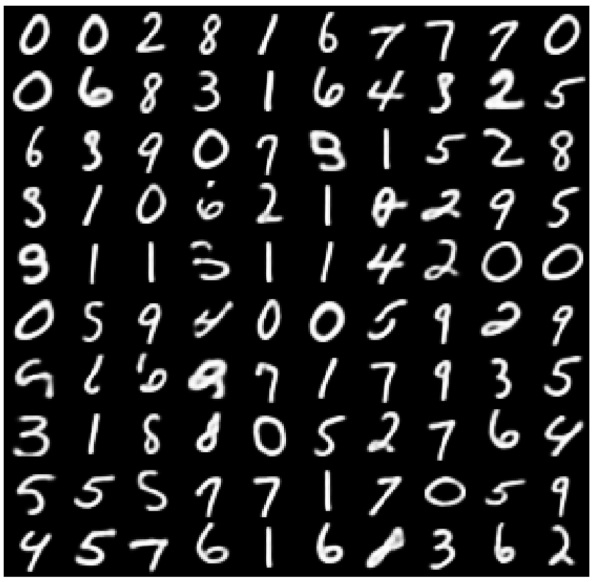


Figure 4: Sample from the DBM with Gibbs = 100.

layer of filters.

When comparing the first layer of the DBM to the second layer, we can see that the filters contain more complex structures, with combinations of loops and some small lines and edges. Most notably, the many of the interconnection weights filter for a circular area in the center of the image, showing the darker area with which the digit would be surrounded.

Samples were then taken using this deep Boltzmann machine (see Figure 4). When compared to the samples taken earlier from the RBM, the quality is markedly better and there is less noise.

Generative Adversarial Networks

In this section we train a deep convolutional generative adversarial network (DCGAN) on the CIFAR dataset. The architectural guidelines for DCGANs set out by Radford & Chintala[4] suggest that:

- Any pooling layers be replaced with strided convolutions (discriminator) and fractional-strided convolutions (generator)
- Use batch normalisation in both the generator and the discriminator
- Remove fully connected hidden layers for deeper architectures
- Use ReLU activation in generator for all layers except for the output, which uses Tanh
- Use LeakyReLU activation in the discriminator for all layers

I chose to use class 3 of the CIFAR dataset (cats), and then trained the DCGAN using the recommendations given above.

GANs and Stability

I struggled to work with DCGAN notebook provided for this assignment – in particular I could not plot the loss or accuracy. Ideally, I would plot the generator and discriminator loss over time, to view the stability of the network. It can be challenging to strain a stable GAN. The reason is that the training process is a dynamical system and inherently unstable, resulting from the simultaneous training of two competing models. The generator model and the discriminator model are trained simultaneously in a 'game', and so any improvement to one model comes at the expense of the other. The ultimate goal of training is therefore to find an equilibrium where the models perform equally well. The concern is that the networks don't converge. GANs will often become stuck in a state where the generator oscillates between generating different kinds of samples.[2] One type of model failure, known as 'mode collapse', occurs when multiple inputs to the generator result in the generation of the same output.

It is also difficult to evaluate whether a GAN is performing well during training. Looking at the loss in the models is not sufficient and often the best metric is a subjective review of the generated images. In the paper "Improved techniques for training GANS", Goodfellow notes

> *Generative adversarial networks lack an objective function, which makes it difficult to compare performance of different models. One intuitive metric of performance can be obtained by having human annotators judge the visual quality of samples.*[6]

Understanding that the loss is not the best metric of model performance during training, it is still interesting to note that the discriminator loss varied between 0.45 and 0.85 during training, and did not converge by the end of training. The generator loss also did not converge, and so the GAN was ultimately unstable.

Optimal Transport and the Wasserstein Metric

In statistics, the Wasserstein metric or Earth Mover's Distance (EMD) is a measure of the distance between two probability distributions over a region D (note that this is simplified, I'm not going to give a formal definition or discuss metric spaces.) Informally, the Wasserstein metric can be interpreted as the minimum cost of transforming one probability distribution into the other, for some defined cost function C. Naturally, this definition is only The above definition is valid only if the two distributions have the same integral.

Optimal Transport and Colour Swapping

One simple application of Wasserstein metric is for the optimal transport (OT) of colour between two images using their normalised colour histograms. Each normalised colour histogram defines a probability distribution. Here we use OT to transfer colour between the two images given in Figure 5.

Figure 5

I'm not going to go into detail about how exactly the optimal transport algorithm works. If you're interested in a gentle introduction to OT and then I'd suggest this video series by Marco Cuturi. The results of the optimal transport are given in Figure 6.

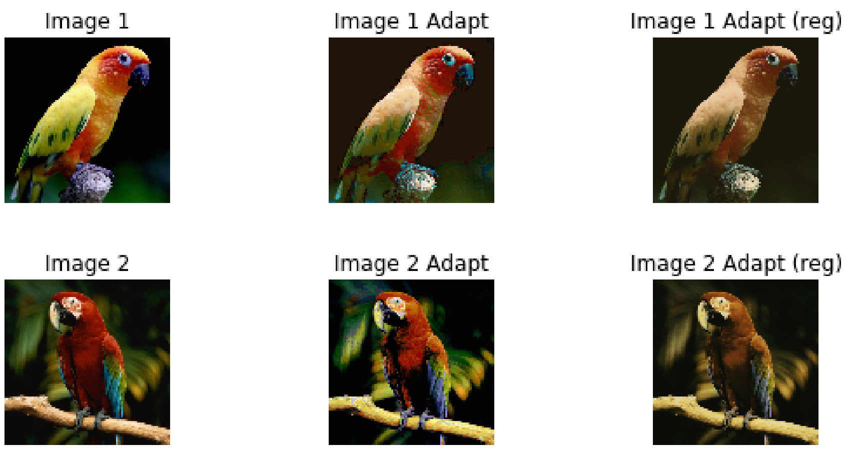


Figure 6: Images with colours swapped using optimal transport.

Looking at the first two columns of images in Figure 6, we can see that the colour histograms of each image have been transformed into the other, but not simply by swapping pixels. For example, note the blue on the wings of the parrot in Image 2. When transporting this colour to Image 1, we want to transport it to the pixel that is closest in terms of colour (assuming there is no other transportation which minimises the cost further). Looking at the adapted version of Image 1, we can see this blue was transported to beak, the pixels of which were blue-purple in the original image.

Wasserstein GANs

Two common divergences used in generative models are the Kullback-Leibler (KL) divergence and the Jensen-Shannon (JS) divergence. These divergences give a measure of 'distance' between two probability distributions. For two normal equivariant probability distributions p and q with mean μ and τ respectively, we can plot the KL and JS divergence as the two distributions as τ moves further from μ. As τ increases, the curve will flatten and therefore the gradient of the divergency will diminish, which means that the generator learns nothing from gradient descent.

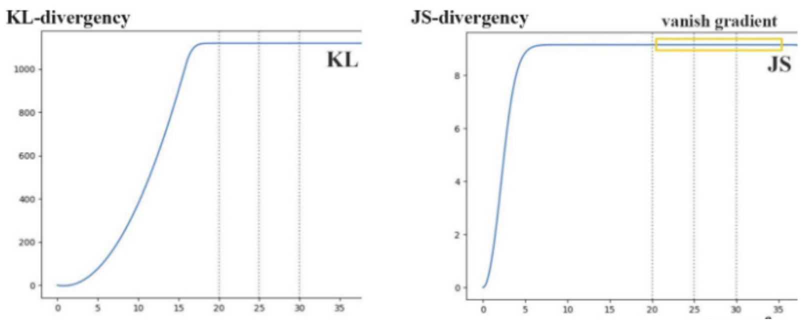


Figure 7: A figure showing the diminishing gradient problem for both the KL and JS divergence. The x-axis is the difference between the τ and μ for the equivariant probability distributions p amd q.

Wasserstein GANs (WGANs) use the Wasserstein metric as a cost function as it has a much smoother gradient which does not diminish as the distributions are separated. This means that there won't be the same vanishing gradient problem and the generator can learn even at the beginning when it is not producing good images and the distributions are very far apart. Without going into the details, the equation for the Wasserstein metric is highly intractable. Using the Kantorovich-Rubinstein duality, the calculation can be simplified to

$$W\left(\mathbb{P}_r, \mathbb{P}_\theta\right) = \sup_{\|f\|_L \leq 1} \mathbb{E}_{x\sim\mathbb{P}_r}[f(x)] - \mathbb{E}_{x\sim\mathbb{P}_\theta}[f(x)]$$

where the function f is a 1-Lipschitz function following a constraint called the Lipschitz constraint. In order to enforce this constraint, Wasserstein GANs apply clipping to restrict the maximum weight values in the discriminator. This is called weight-clipping. However, using weight-clipping to enforce the Lipschitz constraint can lead to problems with convergence and stability. Instead of using weight-clipping, Wasserstein GANs can instead use a gradient penalty to enforce the Lipschitz constraint. Using a gradient penalty makes training more computationally intensive, but can significantly improve the stability of the GAN. As WGANs with gradient penalty are more stable, they can be trained for a much longer time on more powerful networks as we know they will converge.

For this assignment, A standard GAN, a Wasserstein GAN with weight clipping, and a Wasserstein GAN with gradient penalty were each trained on the MNIST dataset. These networks are not convolutional and so performance is not optimal, but we are mostly interested in the relative stability and performance of these different GANs. The standard GAN was unstable and did not converge. The Wasserstein GAN with weight clipping converged. Shown in Figure 8 are the final generated images from the standard GAN and the Wasserstein GAN with weight penalty.

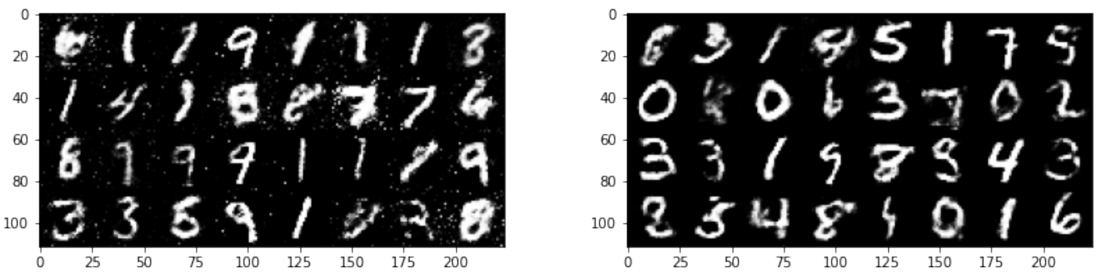


Figure 8: Generated images from the standard GAN *(left)* and Wasserstein GAN with weight penalty *(right)*

References

[1] Fischer, A., and Igel, C. An introduction to restricted boltzmann machines. In *Iberoamerican congress on pattern recognition* (2012), Springer, pp. 14–36.

[2] Goodfellow, I. Nips 2016 tutorial: Generative adversarial networks. *arXiv preprint arXiv:1701.00160* (2016).

[3] Hinton, G. E. A practical guide to training restricted boltzmann machines. In *Neural networks: Tricks of the trade*. Springer, 2012, pp. 599–619.

[4] Radford, A., Metz, L., and Chintala, S. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434* (2015).

[5] Salakhutdinov, R., and Larochelle, H. Efficient learning of deep boltzmann machines. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics* (2010), pp. 693–700.

[6] Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A., and Chen, X. Improved techniques for training gans. In *Advances in neural information processing systems* (2016), pp. 2234–2242.