AN INTRODUCTION TO DEEP GENERATIVE MODELS

A PREPRINT

Elias Baumann
Humboldt University Berlin
elias.baumann@hu-berlin.de

December 14, 2018

ABSTRACT

In this work we give an introduction to deep generative models. These models learn a probability distribution from data or at least are able to learn a sampling process from a distribution. Specifically, this work will first introduce a non-deep model with the restricted Boltzmann machine which is a required base for the deep belief network and deep boltzmann machine. Furthermore, this work will look at current state-of-the-art in deep generative modeling with variational autoencoders and generative adversarial networks. Explanations will be supplemented with exemplary demonstrations on simple datasets.

***K*eywords** Generative Model · Boltzmann machine · Deep Belief Network · Deep Boltzmann Machine · Variational Autoencoder · Generative adversarial network

1 Introduction

Algorithms and models to generate data based on a learned distribution have been around for some time [1]. These generative models are able to generate new, previously unseen, data such as new similar images, sounds and any sort of other data entries in a dataset. Furthermore, deep generative models, models with multiple layers and depth structure are able to accurately model and capture high dimensional probability distributions which is already interesting in itself for mathematical applications. Deep generative models allow for learning of data without the need for explicit labels or goals other then representing the data well. This allows for making use of the large amounts of unlabelled data which have been gathered in the last decade to either build a classifier or regression model or generate even mode data. Semi-supervised approaches based on deep generative models are of high value for companies with a small set of labels which can be used to fine-tune a model which already learned features from a large unlabeled set. Currently the most prominent application against which state-of-the-art models are evaluated against is the generation of realistic high resolution images and all varieties of the same fundamental goal [10][15]. However another interesting application is the generation of synthetic data to expand a previously small dataset such that other deep learning algorithms can build on the generated large set[4]. This work will introduce several types of deep generative models explaining how each model works in detail and which issues each model faces in order to also demonstrate why current research focuses on certain generative models more than on others.

2 Restricted Boltzmann Machines

Restricted Boltzmann Machines (RBMs) are restricted versions of Boltzmann machines, which in essence are Networks of stochastic units similar to neural networks [1]. They are able to model a distribution $p(v)$ over input data v by representing any input data vector using hidden units. Even if the distribution of the input is unknown such as in cases where input data consists of very complex images, Boltzmann machines are able to capture a distribution over all samples. RBMs fall under the category of energy-based models meaning that it is possible to evaluate the current state of the model using an energy function. In order to learn a distribution, boltzmann machines adjust parameters for every unit (biases) and edge (weights). However, without the restrictions of the RBM, learning such markov random field is time consuming and difficult [6].

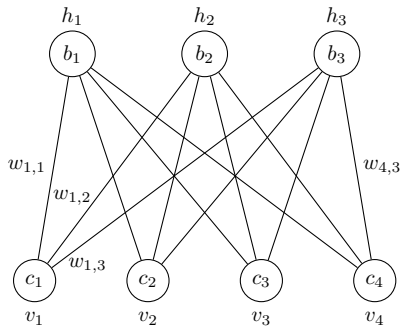


Figure 1: Restricted Boltzmann Machine

2.1 Training

Therefore, Ackley et al. (1985) [1] introduced an algorithm to train a restricted version of the boltzmann machine. The restricted boltzmann machine consists of two layers. A visible layer $\boldsymbol{v}$ with visible units v_i and a hidden layer $\boldsymbol{h}$ with hidden units h_i. The visible units correspond to observations from the input dataset and the hidden units model dependencies between the hidden units. Every hidden unit is connected to every visible unit and vise versa with continuous weights $w_{i,j}$ and every unit has a continuous activation threshold (bias). The number of hidden units can be chosen freely, yet commonly a lower number of units is used then the number of input units. In the restricted boltzmann machine there are no connections between units of the same layer. This property will become important, when a training algorithm is considered. In this initial overview, the binary restricted boltzmann machine is considered where both hidden and visible units take on binary values. Figure 1 shows the basic structure of a restricted boltzmann machine with biases b, c. The restricted boltzmann machine can be described by the following energy function:

$$E(\boldsymbol{v}, \boldsymbol{h}) = -\boldsymbol{h}^T \boldsymbol{W} \boldsymbol{v} - \boldsymbol{c}^T \boldsymbol{v} - \boldsymbol{b}^T \boldsymbol{h} \tag{1}$$

The energy function can also be expressed as sums over all units within a layer respectively. As an rbm is a undirected graphical model and can be expressed using the gibbs distribution [20]:

$$p(\boldsymbol{v}, \boldsymbol{h}) = \frac{1}{Z} e^{-E(\boldsymbol{v}, \boldsymbol{h})} \tag{2}$$

which incorporates the normalization constant (also known as partition function) Z:

$$Z = \sum_{\boldsymbol{v}, \boldsymbol{h}} e^{-E(\boldsymbol{v}, \boldsymbol{h})} \tag{3}$$

Z is used to normalize the result such that the sum over the distribution is guaranteed to be one. However, considering that restricted boltzmann machines usually do not consist of just seven units as in Figure 1, the time required to sum over all possible states of the model increases exponentially with the number of units. The partition function Z is thus intractable making a calculation of $p(v)$ intractable as well [9]. Considering the task of modeling the probability of input data and the generating data similar to the input, the conditional distributions $p(\boldsymbol{h}|\boldsymbol{v})$ and $p(\boldsymbol{v}|\boldsymbol{h})$ within the model might be more interesting than the combined distribution $p(\boldsymbol{v}, \boldsymbol{h})$. Given the structure of a RBM, where units within a layer are not connected, every unit in a layer is conditionally independent of all other units in the layer given the other layer. This holds true in particular because of the local markov property which proposes that any node in an undirected graphical model is is conditionally independent of all other nodes given its markov blanket. The markov blanket refers to the set of all immediate neighbours of a node [20]. Using this property and the proof of the Hammersley-Clifford theorem [7] the conditional distributions over all unit can factorize into positive functions and can be written as follows:

$$p(\boldsymbol{h}|\boldsymbol{v}) = \prod_j p(h_j|\boldsymbol{v}) \tag{4}$$

And the probability of an individual unit is defined by:

$$(h_i = 1|\boldsymbol{v}) = \sigma(\boldsymbol{W}_{j,*}\boldsymbol{v} + bj) \tag{5}$$

where $\sigma(x) = \frac{1}{1+exp(-x)}$. Equivalent equations can be derived for $p(\boldsymbol{v}|\boldsymbol{h})$ [6]. The training process will make use of both conditional probabilities to evaluate and update the model and its parameters . The general training algorithm to

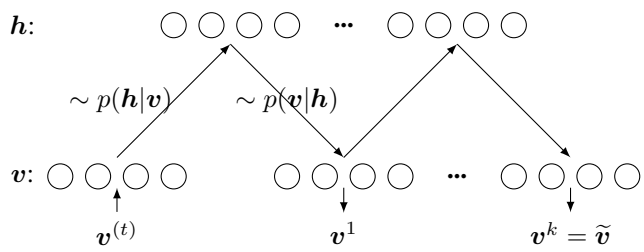


Figure 2: Gibbs sampling in a restricted Boltzmann machine

adjust the set of parameters $\theta = \{\boldsymbol{W}, \boldsymbol{b}, \boldsymbol{c}\}$ will make use of maximum likelihood methods. In this particular case the negative log-likelihood of $p(\boldsymbol{v})$ is minimized over all samples of the dataset T:

$$\min_{\theta} \frac{1}{T} \sum_{t} -log(p(\boldsymbol{v}^{(t)})) \tag{6}$$

Updating parameters to find a minimum is done via stochastic gradient descent (SGD). Therefore the partial derivatives in regard to every paramter θ have to be taken of the negative log-likelihood. In the example of a restricted boltzmann machine, the loglikelihood can be reformulated to a difference between two expectations:

$$\frac{\partial - logp(\boldsymbol{v}^{(t)})}{\partial \theta} = \mathbb{E}_{\boldsymbol{h}}\left[\frac{\partial E(\boldsymbol{v}^{(t)}, \boldsymbol{h})}{\partial \theta}\middle|\boldsymbol{v}^{(t)}\right] - \underbrace{\mathbb{E}_{\boldsymbol{v},\boldsymbol{h}}\left[\frac{\partial E(\boldsymbol{v}, \boldsymbol{h})}{\partial \theta}\right]}_{\text{Intractable}} \tag{7}$$

2.2 Contrastive Divergence

Running Markov chains to convergence is time intensive, which leads to the introduction of an algorithm called contrastive divergence (CD) [12]. K-step Contrastive divergence shows that the approximation of the models expectation can already be reached with a low number of Gibbs Sampling steps where Gibbs sampling is in essence a Markov chain. In practice the number of steps required to get a sufficiently accurate estimate is $k = 1$. Gibbs sampling works by alternatingly passing a sample or a random initialization up and down the model using Equation 4 and its corresponding counter equation for $p(\boldsymbol{v}|\boldsymbol{h})$. As the equation returns of probabilities, it has to be sampled from to get an allocation for the units of a layer. The conditional independence also allows for parallel execution of each Gibbs sampling step over all units within a layer, making this in essence two matrix multiplication steps in order to get a reasonable model estimate. Figure 2 demonstrates the process of Gibbs sampling for k steps using a data sample as starting values [6]. A further improvement on CD-k can be done by using just one permanently existing Gibbs chain to sample from instead of initializing new chains for every parameter update step. This persistent contrastive divergence approach requires a learning rate α to decrease over time (e.g. by using $\alpha = \frac{1}{t}$ such that the chain stays close to the models distribution. To get a sample from the chain, a low number of Gibbs sampling iterations are performed [30]. Using the point estimate to get a model expectation, partial derivatives in regard to parameters can be computed. Using matrix notation updates are performed using the Equations 8,9,10.

$$\boldsymbol{W} = \boldsymbol{W} + \alpha\Big(\sigma(\boldsymbol{W}\boldsymbol{v}^{(t)} + \boldsymbol{b})\boldsymbol{v}^{(t)^T} - \sigma(\boldsymbol{W}\widetilde{\boldsymbol{v}} + \boldsymbol{b})\widetilde{\boldsymbol{v}}^T\Big) \tag{8}$$

$$\boldsymbol{b} = \boldsymbol{b} + \alpha\Big(\sigma(\boldsymbol{W}\boldsymbol{v}^{(t)} + \boldsymbol{b}) - \sigma(\boldsymbol{W}\widetilde{\boldsymbol{v}} + \boldsymbol{b})\Big) \tag{9}$$

$$\boldsymbol{c} = \boldsymbol{c} + \alpha\Big(\boldsymbol{v}^{(t)} - \widetilde{\boldsymbol{v}}\Big) \tag{10}$$

Model updates are repeated over all training examples and are often combined in mini-batches where multiple updates are meaned over and applied afterwards. It is also common to iterate over the same dataset for many epochs (One epoch is a full iteration over the entire input set) [20]. After training, the RBM with its learned parameters can be used to get a sample from the model distribution. By using a sample from a test set, an a similar output can be computed using a few steps of Gibbs sampling.

2.3 Gaussian Bernoulli RBMs

So far, only the case where both hidden and visible units could only have binary states was considered. However in many cases, one wants to model real valued data (i.e. images). A Gaussian-Bernoulli restricted Boltzmann machine

(GBRBM) was proposed by Welling et al. (2005) [31] where the visible units are assumed to be of Gaussian distribution. The distributions mean is calculated using hidden units and the variance can either be a learned parameter or be set to a fixed value. In this explanation, variance will be set to one. This leads to Equation 11 as the visible units new conditional probability.

$$p(\boldsymbol{v}|\boldsymbol{h}) = \mathcal{N}(\boldsymbol{W}\boldsymbol{h} + \boldsymbol{b}, 1) \tag{11}$$

The conditional probability $p(\boldsymbol{h}|\boldsymbol{v})$ is unaffected by this change. However the energy function of the model also changes as the quadratic term of the Gaussian has to be included (Equation 12) [19].

$$E(\boldsymbol{v}, \boldsymbol{h}) = \frac{(\boldsymbol{v} - \boldsymbol{b})^2}{2} - \boldsymbol{W}\boldsymbol{v}\boldsymbol{h} - \boldsymbol{c}\boldsymbol{h} \tag{12}$$

As for the regular restricted Boltzmann machine, parameter updates can be derived and the machine can be now trained using real valued inputs.

2.4 Applying a RBM

In order to demonstrate the capabilities of a RBM, we implement a binary restricted Boltzmann machine using Python and trained the model on MNIST [17] which is a dataset of 60000 handwritten digits with all digits from zero to nine and stored as 28×28 grey scale images. We re-scale the images to be binary (i.e. consisting only of zeros and ones) to be able to use a binary RBM. The model was initialized with 1000 hidden units, a learning rate of 0.01 and was trained for 20 epochs with a batch size of 20. Figure 3 shows results of the model after 5 steps of Gibbs-sampling for different input images.

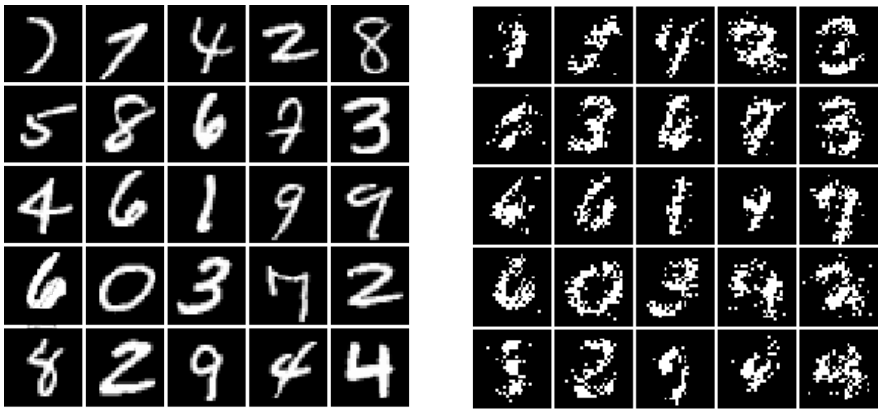


Figure 3: Samples from restricted Boltzmann machines (left input, right result)

Considering that the topic of this work is deep generative models, the following two chapters explain two different approaches to deal with larger and more complex data by stacking and combining multiple restricted Boltzmann machines

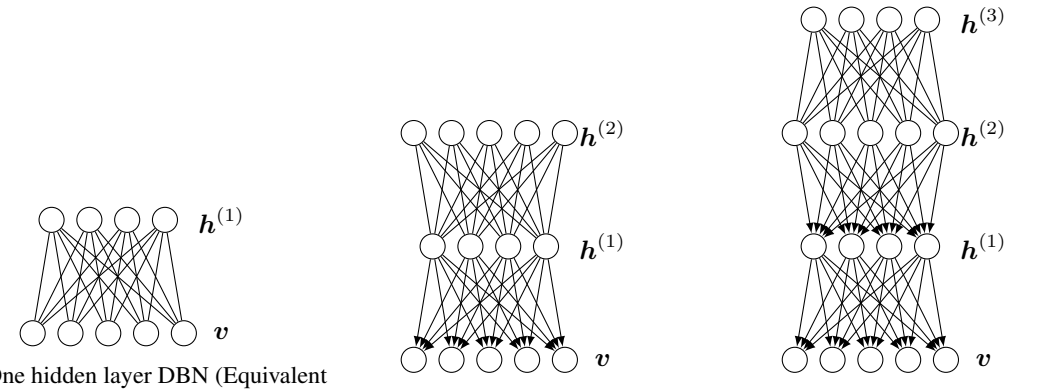
3 Deep Belief Networks

Originally introduced by Hinton et al. (2006) [13], deep belief networks (DBN) are an approach to stack multiple restricted Boltzmann machines to be able to model more complex distributions. Figure 4c shows a network graph of a deep belief network with three hidden layers. In a DBN, only the top two hidden layers are connected via undirected edges making them a RBM. All lower layers form sigmoid belief networks, feed forward networks with directed edges pointing in the direction of the visible layer. In the following explanations, the same notation will be used as for the restricted Boltzmann machine and all explanations make use of binary units to simplify formulas and explanations. However a deep belief network can also be constructed using Gaussian units.

A deep belief network is initialized using a restricted Boltzmann machine making a one layer deep belief network equivalent to an RBM (Figure 4a). The probability distribution of input data can be represented by the RBM by summing over all possible states of hidden units of the combined distribution. This notation in Equation 13 will be used to explain the combination of multiple layers.

$$p(\boldsymbol{v}) = \sum_{\boldsymbol{h}^{(1)}} p(\boldsymbol{v}, \boldsymbol{h}^{(1)}) \tag{13}$$

The training of this initial restricted Boltzmann machine can be done using the same methods as explained in Section 2. However $p(\boldsymbol{v})$ might be very difficult to calculate such that instead of calculating the value, it is approximated using the variational lower bound [25]. The variational bound approximation will later also help to ensure that additional layers improve on the log likelihood.



(a) One hidden layer DBN (Equivalent to RBM)

(b) Two hidden layer DBN

(c) Three hidden layer RBM

Figure 4: Stacking RBMs to form a DBN

3.1 Training using variational Lower Bound

The variational lower bound allows for an approximation of $p(\boldsymbol{h}^{(1)}|\boldsymbol{v})$ using a distribution $q(\boldsymbol{h}^{(1)}|\boldsymbol{v})$ which can be chosen to be factorizing. An important part of the variational lower bound approximation is Jensen's Inequality. The inequality is defined on convex functions, but also holds on concave functions such as the log function. Equation 14 holds true and can intuitively be understood by looking at Figure 5. By moving weights w_i which sum up to 1 out of the log, we get a lower bound approximation which is a diagonal far from the actual log but when considering more than two points becomes more accurate.

$$log(\sum_i w_i a_i) \geq \sum_i w_i log(a_i), \qquad \sum_i w_i = 1, w_i > 0 \tag{14}$$

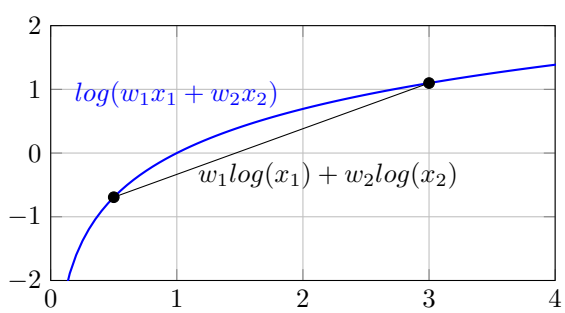


Figure 5: Jensen's inequality on concave log

In particular, this inequality can be used for probabilities and conditional probabilities as well. Here the difference between the approximation $q(\boldsymbol{h}^{(1)}|\boldsymbol{v})$ and the actual $log(p(\boldsymbol{v}))$ is the Kullback-leibler Divergence (Equation 15) which is 0 when p and q are equal and > 0 otherwise.

$$KL(q||p) = \Sigma_{\boldsymbol{h}^{(1)}} q(\boldsymbol{h}^{(1)}|\boldsymbol{v}) log\Big(\frac{q(\boldsymbol{h}^{(1)}|\boldsymbol{v})}{p(\boldsymbol{h}^{(1)}|\boldsymbol{v})}\Big) \tag{15}$$

Equations 16,17,18,19 show how to reformulate the original problem of Equation 13 using Jensen's Inequality on the log probability in Equation 18 to move the approximate distribution q out of the log.

$$log(p(\boldsymbol{v})) = log\left(\sum_{\boldsymbol{h}^{(1)}} p(\boldsymbol{v}, \boldsymbol{h}^{(1)})\right) \tag{16}$$

$$= log\left(\sum_{\boldsymbol{h}^{(1)}} q(\boldsymbol{h}^{(1)}|\boldsymbol{v}) \frac{p(\boldsymbol{v}, \boldsymbol{h}^{(1)})}{q(\boldsymbol{h}^{(1)}|\boldsymbol{v})}\right) \tag{17}$$

$$\geq \sum_{\boldsymbol{h}^{(1)}} q(\boldsymbol{h}^{(1)}|\boldsymbol{v}) log\left(\frac{p(\boldsymbol{v}, \boldsymbol{h}^{(1)})}{q(\boldsymbol{h}^{(1)}|\boldsymbol{v})}\right) \tag{18}$$

$$= \sum_{\boldsymbol{h}^{(1)}} q(\boldsymbol{h}^{(1)}|\boldsymbol{v}) log(p(\boldsymbol{v}, \boldsymbol{h}^{(1)}) - \sum_{\boldsymbol{h}^{(1)}} q(\boldsymbol{h}^{(1)}|\boldsymbol{v}) log(q(\boldsymbol{h}^{(1)}|\boldsymbol{v})) \tag{19}$$

The second part of Equation 19 is equivalent to the entropy H of $q(\boldsymbol{h}|\boldsymbol{v})$. When training a DBN the initial approximation of $q(\boldsymbol{h}^{(1)}|\boldsymbol{v})$ is estimated using samples from the training dataset. An optimal RBM yields a q which is equal to p making the entire term an equality. However, deep belief networks are created by stacking RBMs. In Equation 19, a second RBM will improve on the first part of the equation, particularly the estimation of $p(\boldsymbol{v}, \boldsymbol{h}^{(1)})$ which can be split up into two parts (Equation 20).

$$log(p(\boldsymbol{v}, \boldsymbol{h}^{(1)})) = log(p(\boldsymbol{v}|\boldsymbol{h}^{(1)}) + log(p(\boldsymbol{h}^{(1)})) \tag{20}$$

The latter of which will be used as the estimation goal for the second RBM which is defined by Equation 21. The improvement of a DBN over a simple RBM can therefore only happen in this parameter of the equation.

$$p(\boldsymbol{h}^{(1)}) = \sum_{\boldsymbol{h}^{(2)}} p(\boldsymbol{h}^{(1)}, \boldsymbol{h}^{(2)}) \tag{21}$$

To guarantee an improvement over the log likelihood of the RBM the weights of the first RBM are held fixed but are used as transposed initialization for the second layer RBM. By fixing parameters the second RBM optimizes Equation 22 which is in essence an RBM with input samples from $q(\boldsymbol{h}^{(1)}|\boldsymbol{v})$.

$$\sum_{\boldsymbol{h}^{(1)}} q(\boldsymbol{h}^{(1)}|\boldsymbol{v}) log(p(\boldsymbol{h}^{(1)})) \tag{22}$$

The guaranteed improvement comes from using the transposed Weights ${W^{(1)}}^T$, which means that the second RBM is an inverse of the first and combined with a sigmoid belief network will output the exact same values as the single RBM [13]. Training the second layer will then increase the lower bound as the estimation of $p(\boldsymbol{h}^{(1)}|\boldsymbol{v})$ is already accurate. To train the second layer RBM, samples from $\boldsymbol{h}^{(1)}$ are required. To obtain them, an bottom up pass of all samples through the first RBM is performed which yields the same number of probability vectors. These can then be used to sample data for $\boldsymbol{h}^{(1)}$. The same procedure can then be repeated for more layers, however there is no guarantee to improve on the log-likelihood [25]. When generating samples for higher level RBMs, one can rely on the dataset generated for the lower level RBM and again pass it up and sample from distributions. Finally, this layer-wise algorithm will return a trained deep belief network which can be used to generate data or as weight initialization for a classifier. Samples from this network can be generated by randomly initializing $h^{(L-1)}$ (Second to last hidden layer), or using a bottom up pass of a sample to initialize the same layer. Then the gibbs sampling steps known from RBM training are performed for a number of iterations and the final result is passed down to the visible units [9].

3.2 Up-Down Algorithm

This general training algorithm does not include combined optimization of all layers, which is why a variant of the wake-sleep algorithm is introduced by Hinton et al. (2006) [13]. The algorithm requires untying top-down and bottom up weights in all directed edges and optimizing them separately.

1. Samples are bottom-up passed through the network using bottom up weights and activating units depending on probability. For every layer, the connected parent hidden units are then optimized using the same maximum likelihood optimization as for training an RBM.
2. The top two layers are trained just like a regular RBM using the same passed up samples.

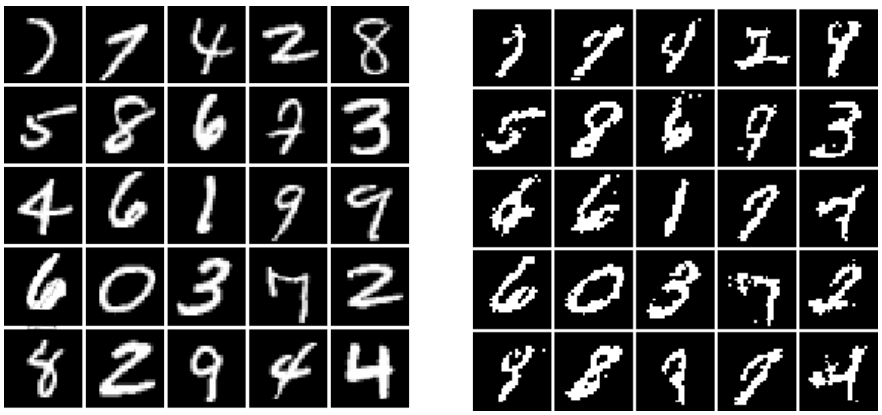


Figure 6: Results of DBN sampled for 5 steps (left input, right result

3. After Gibbs Sampling from the top level RBM for a number of steps, the acquired samples are used for a top-down pass. Similar to the first step, the bottom-up weights are now adjusted to construct the higher level input.

Of course, penalties and learning rates can be introduced to influence the behavior of this joint optimization procedure [13]. This algorithm also alleviates the problems of "Explaining away" when training a deep belief network as during training, top down inference does not depend on all layers but instead on the hidden layer of a restricted Boltzmann machine. The phenomenon of explaining away makes parent nodes in a directed graph dependent as both can be cause for the probability of the child node but the confirmation of one makes the other less likely [2].

3.3 Applying a DBN

To demonstrate the capabilities of a deep belief network, we implement an architecture introduced by Hinton et al. (2006) [13] using the MNIST dataset. As a RBM base, we use a fast implementation of Scikit learn [23] and construct a three hidden layer DBN. We disregard using the transpose of $W^{(1)}$ and therefore define the hidden units to be 500,500 and 1000 respectively. All RBMs are pre-trained using the same parameters as for the single RBM trained before. We do not implement the up-down algorithm, yet still yield significantly better results than by using the RBM only. However it should be noted that training time is also more than tripled. Figure 6 shows results the same input grid as for the RBM but with the DBN reconstructions.

The deep belief network is somewhat limited by the directed edges when generating data as lower level information is not considered when inferring higher level data. Deep Boltzmann machines solve this problem and provide a fully undirected model with multiple layers.

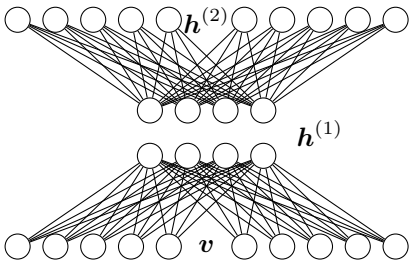
4 Deep Boltzmann Machines

Deep Boltzmann Machines (DBM) are undirected graphical models with conditional independence between layers. In particular, DBMs are very similar to deep belief networks yet they overcome the restriction of directed edges. Figure 7b shows the graphical model of a deep Boltzmann machine with two hidden layers. DBMs allow for learning increasingly complex representations such as distributions of speech and objects in images while also making it possible to include top-down information during the approximate inference step [26].

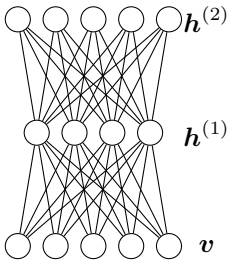
4.1 Training a binary DBM

The following explanation again will consider the case of binary units for all layers and will explain the workings of a two layer deep Boltzmann machine such as shown in Figure 7b. Given that a deep Boltzmann machine consists of multiple layers of hidden units, it also has multiple Weight matrices $\{\boldsymbol{W}^{(1)}, \boldsymbol{W}^{(2)}\}$ and biases $\{\boldsymbol{b}, \boldsymbol{c}^{(1)}, \boldsymbol{c}^{(2)}\}$ for every layer. Accordingly the Energy function is adjusted to incorporate the multiple layers, while the joint probability of all layers can still be expressed just like in Equation 2.

$$E(\boldsymbol{v}, \boldsymbol{h}^{(1)}, \boldsymbol{h}^{(2)}) = -\boldsymbol{v}^T \boldsymbol{W}^{(1)} \boldsymbol{h}^{(1)} - \boldsymbol{h}^{(1)^T} \boldsymbol{W}^{(2)} \boldsymbol{h}^{(2)} - \boldsymbol{h}^{(1)^T} \boldsymbol{b}^{(1)} - \boldsymbol{h}^{(2)^T} \boldsymbol{c}^{(2)} - \boldsymbol{v}^T \boldsymbol{b} \tag{23}$$



(a) Pre-training RBMs with double inputs in bottom and top most layer



(b) Deep Boltzmann machine with 2 hidden layers

Initialization in DBMs happens similar to DBNs where multiple restricted Boltzmann machines are pre-trained and stacked. However, in a DBM the goal is a model with undirected edges. To get such a model, the double counting problem has to be solved [27]. When constructing the model stack in a DBN, $p(\boldsymbol{h}^{(1)})$ is initially estimated by the lowest RBM, but then replaced in the following step by $p(\boldsymbol{h}^{(1)}; \boldsymbol{W}^{(2)}$ (See Equation 22). With undirected edges, $p(\boldsymbol{h}^{(1)}; \boldsymbol{W}^{(1)}, \boldsymbol{W}^{(2)})$ can be estimated by averaging over both models of $\boldsymbol{h}^{(1)}$ using $\frac{1}{2}\boldsymbol{W}^{(1)}$ and $\frac{1}{2}\boldsymbol{W}^{(2)}$. If weights are not halved, information stemming from $\boldsymbol{v}$ will be counted twice as the second RBM also depends on the original input. Salakhutdinov and Hinton (2009) [26] therefore propose to double the input layer for the first RBM and top most layer for the last RBM during pre-training as well as to double weights for all intermediate RBMs not directly connected to the top or bottom layers. When combining a model from only two RBMs, inference of $\boldsymbol{h}^{(1)}$ then receives half weights bottom-up and half weights top-down and can be computed using Equation 24. Figure 7a graphically shows the double inputs for top and bottom layer.

$$(h_j^{(1)} = 1|\boldsymbol{v}, \boldsymbol{h}^{(2)}) = \sigma\left(\boldsymbol{W}_{*,j}^{(1)}\boldsymbol{v} + \boldsymbol{W}_{j,*}^{(2)}\boldsymbol{h}^{(2)} + \boldsymbol{c}^{(1)}\right) \tag{24}$$

Using this initialization, the joint optimization over all layers converges faster.

Joint optimization happens again via maximum likelihood learning over parameters $\theta = \{\boldsymbol{W}^{(1)}, \boldsymbol{W}^{(2)}, \boldsymbol{b}^{(1)}, \boldsymbol{b}^{(2)}, \boldsymbol{c}\}$ using Equation 25 which is similar to the same equation for RBMs, however both the data-dependent part as well as the data-independent part of the equation are intractable as summing over all combined states of multiple layers would be necessary.

$$\frac{\partial - logp(\boldsymbol{v}^{(t)})}{\partial\theta} = \underbrace{\mathbb{E}_{\boldsymbol{h}^{(1)},\boldsymbol{h}^{(2)}}\left[\frac{\partial E(\boldsymbol{v}^{(t)}, \boldsymbol{h}^{(1)}, \boldsymbol{h}^{(2)})}{\partial\theta}\middle|\boldsymbol{v}^{(t)}\right]}_{\text{Data-dependent}} \underbrace{-\mathbb{E}_{\boldsymbol{v},\boldsymbol{h}^{(1)},\boldsymbol{h}^{(2)}}\left[\frac{\partial E(\boldsymbol{v}, \boldsymbol{h}^{(1)}, \boldsymbol{h}^{(2)})}{\partial\theta}\right]}_{\text{Data-independent}} \tag{25}$$

Therefore, both expectations have to be approximated.

4.1.1 Estimating the data-independent expectation

The data-independent estimation can be done using a similar approach as for RBMs as one can make use of the independence of two layers separated by another intermediate layers i.e. in a three layer deep Boltzmann machine $\boldsymbol{v}$ and $\boldsymbol{h}^{(2)}$ are independent conditional on $\boldsymbol{h}^{(1)}$. Therefore the entire network can be collapsed to a bipartite graph of which a point estimate can be drawn using the Gibbs sampling approach from restricted Boltzmann machines. Figure 8 demonstrates the sampling process for a three layer deep Boltzmann machine, however the process can be arbitrarily extended for more layers.

In deep Boltzmann machines the initial values for the Gibbs chain are randomly initialized and results are averaged over multiple runs with different random initializations. Multiple persistent chains will be used for this purpose [27].

4.1.2 Estimating the data-dependent expectation

The data-dependent expectation now also has to be approximated but in this case using mean-field inference. The variational bound will be used again to get an approximate of a conditional distribution $p(\boldsymbol{h}^{(1)}, \boldsymbol{h}^{(2)}|\boldsymbol{v}; \theta)$ with a fully factorized distribution $q(\boldsymbol{h}^{(1)}, \boldsymbol{h}^{(2)}|\boldsymbol{v}; \boldsymbol{\mu})$. Equation 26 shows the modified version including multiple hidden layers.

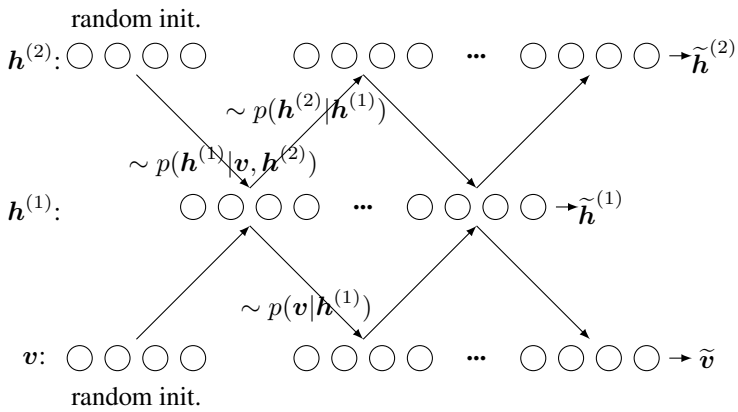


Figure 8: Gibbs sampling in a three layer deep Boltzmann machine

$\mathcal{H}(q)$ refers to the entropy of $q(\boldsymbol{h}^{(1)}, \boldsymbol{h}^{(2)}|\boldsymbol{v};\boldsymbol{\mu})$

$$log(p(\boldsymbol{v})) \geq \sum_{\boldsymbol{h}^{(1)},\boldsymbol{h}^{(2)}} q(\boldsymbol{h}^{(1)}, \boldsymbol{h}^{(2)}|\boldsymbol{v};\boldsymbol{\mu}) log(p(\boldsymbol{h}^{(1)}, \boldsymbol{h}^{(2)}, \boldsymbol{v})) + \mathcal{H}(q) \tag{26}$$

Using the naive mean field approximation for q the distribution can be rewritten as a product where the probability of a single unit being active is given by mean-field parameters $\boldsymbol{\mu}$ (Equation 27).

$$q(\boldsymbol{h}|\boldsymbol{v};\boldsymbol{\mu}) = \prod_j \prod_k q(h_j^1) q(h_k^2), \quad q(h_i^l = 1) = \mu_i^l \tag{27}$$

Given that $q(h_i^l = 1) = \mu_i^l$ and using the information that the energy in a Boltzmann machine is equivalent to the negative log of the probability [29], the log probability can be rewritten to include the terms from the energy function from Equation 1 such that optimizing the entire term for optimal $\boldsymbol{\mu}$ can be done with simple operations (Equation 28).

$$logp(\boldsymbol{v};\theta) \geq \boldsymbol{v}^T \boldsymbol{W}^{(1)} \boldsymbol{\mu}^{(1)} + \boldsymbol{\mu}^{(1)^T} \boldsymbol{W}^{(2)} \boldsymbol{\mu}^{(2)} + +\boldsymbol{\mu}^{(1)^T} \boldsymbol{b}^{(1)} + \boldsymbol{\mu}^{(2)^T} \boldsymbol{c}^{(2)} + \boldsymbol{v}^T \boldsymbol{b} - log\mathcal{Z} + \mathcal{H}(q) \tag{28}$$

In order to optimize the parameters μ every per layer $\mu^{(l)}$ is updated layer after layer using mean-field fixed point equations (Equations 29,30)

$$\boldsymbol{\mu}^{(1)} = \sigma(\boldsymbol{v}^T \boldsymbol{W}^{(1)} + \boldsymbol{W}^{(2)} \boldsymbol{\mu}^{(2)} + \boldsymbol{b}^{(1)}) \tag{29}$$

$$\boldsymbol{\mu}^{(2)} = \sigma(\boldsymbol{\mu}^{(2)^T} \boldsymbol{W}^{(3)} + \boldsymbol{b}^{(1)}) \tag{30}$$

In this update process, the distributions are not sampled from, rather the probability vectors $\boldsymbol{\mu}$ are used directly. The process is repeated multiple times over random initializations of $\boldsymbol{\mu}$ and again averaged per update.

4.1.3 Parameter updates

Using both processes, the MCMC based inference and the approximation to the log lower bound, model parameters of the DBM can be updated e.g. for $\boldsymbol{W}^{(1)}$ by using Equation 31 where $\boldsymbol{\mu}^{(1)}$ is the parameter from the fixed point equations and $\widetilde{\boldsymbol{v}}, \widetilde{\boldsymbol{h}}^{(1)}$ are point estimates from Gibbs sampling.

$$\boldsymbol{W}^{(1)} = \boldsymbol{W}^{(1)} + \alpha \left(\frac{1}{N} \sum_{n=1}^{N} \boldsymbol{v}_n (\boldsymbol{\mu}_n^{(1)})^T - \frac{1}{M} \sum_{m=1}^{M} \widetilde{\boldsymbol{v}}_m (\widetilde{\boldsymbol{h}}_m^{(1)})^T \right) \tag{31}$$

N, M are hyper-parameters indicating over how many samples averages should be taken for each estimation method. To improve convergence for this algorithm the damping coefficient α has to decrease over time e.g. by defining it as $1/t$ of T samples [32]. The trained deep Boltzmann machine can then be used to generate samples from the learned distribution by randomly initializing all units and using Gibbs Sampling to get a model estimate. In practice, it is common to do gibbs sampling over multiple thousand steps. Salakhutdinov et al. 2009 use 100.000 steps of gibbs sampling to generate MNIST derived images[26]. Results of their approach can be found in figure 9

So far, only approaches based on the restricted Boltzmann machine have been discussed. However, apart from the RBM itself, none of the approaches discussed are commonly used in current literature. Instead two different types of deep generative models are used: Variational autoencoders and generative adversarial networks, both of which are based on feed-forward artificial neural networks.

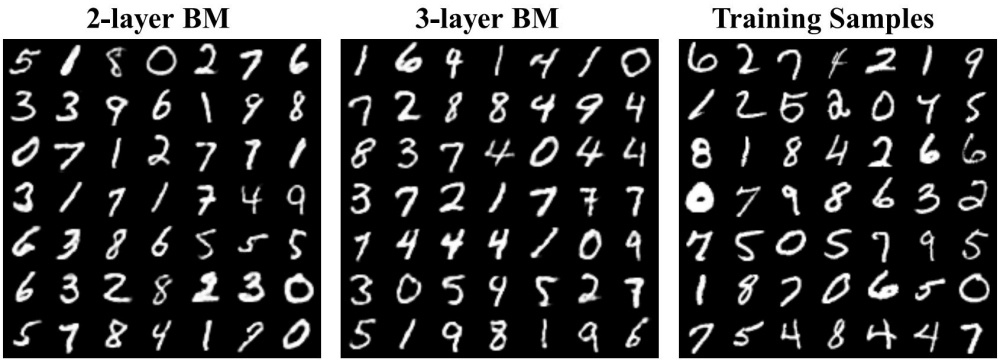


Figure 9: Deep Boltzmann Machine on MNIST samples ([26])

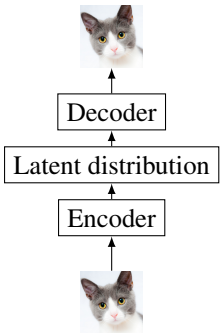


Figure 10: Basic model of the VAE

5 Variational Autoencoder

Variational Autoencoders try to model the underlying distribution of input data but without the computationally expensive MCMC based approaches [15],[24]. Instead they rely on standard function approximators such as feed-foward neural networks which can be trained using only stochastic gradient descent. This approach relies on the fact that large amounts of data are available for training, in particular for unsupervised training on images, and given a sufficiently large model, the distribution approximation should only have small errors [5]. Figure 10 shows the basic architecture of the VAE. The VAE models a distribution of the input data and allows for sampling from that distribution such that the only real overlap with autoencoders is the encoder-decoder structure as well as the unsupervised learning structure of comparing input to output. In the following, all variables can be considered vectors unless indicated otherwise. The variational auto encoder is defined by an input vector x, a latent space z with probability density function $p(\boldsymbol{z})$ and a decoder with input $z \sim p(z)$ of which the goal is to generate samples close to x. The entire model maximizes the likelihood of $p(x)$. Given the latent distribution $p(z)$, $p(x)$ can be rewritten as an integral over all configurations of the latent space (Equation 32)

$$p(x) = \int p(x|z)p(z)dz \tag{32}$$

where $p(x|z)$ can be approximated using a neural network or any other differentiable function approximator and $p(z) = \mathcal{N}(0, I)$. Given that an integral over all possible configurations of latent variables is intractable for a complex model, Equation 32 is intractable. In the same way the posterior $p(z|x)$ is intractable as well as it contains $p(x)$ (Equation 33) [5].

$$p(z|x) = p(x|z)p(z)/ \underbrace{p(x)}_{\text{intractable}} \tag{33}$$

5.1 Training VAEs

Similar to DBNs and DBMs, the distribution of $p(x)$ can be approximated using the variational lower bound and a $q(z|x)$ can be defined using an encoder network to approximate $p(z|x)$ for the bound. Equation 34 splits the bound approximation into two essential parts for the VAE. The decoder will attempt to maximize the Expectation of $log(p(x|z))$ using samples z from distribution $q(z|x)$. The encoder on the other hand will try to minimize the expectation difference

between the distributions $q(z|x)$ and $p(z)$. This difference is equivalent to the Kullback-Leibler divergence which is zero, in case distributions exactly match and up to one if they do not overlap [16]. As $p(z)$ was previously defined as a normal distribution $p(z) = \mathcal{N}(0, I)$, $q(z|x)$ will converge towards the same normal distribution given samples of x.

$$log(p(x)) \geq \mathbb{E}_{z\sim q(z|x)}[log(p(x|z))] - \underbrace{\mathbb{E}_{z\sim q(z|x)}[log(\frac{q(z|x)}{p(z)})]}_{\mathcal{D}_{\mathcal{KL}}[q(z|x)||p(z)]} \tag{34}$$

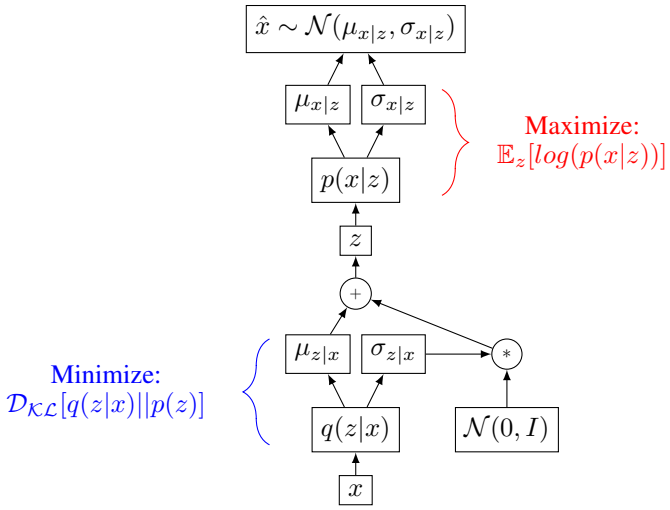
To learn to minimize and maximize the terms of the equation, q and p will be parametrized with ϕ and θ. Then, differentiating according to the parameters, the model can get a learning gradient for stochastic gradient descent.

5.1.1 reparametrization Trick

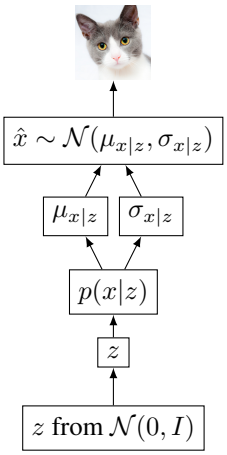
There is an issue however which has not been discussed so far. The model includes a latent distribution which would have to be included in the partial derivation but distributions are not differentiable. Therefore, the VAE uses an approach now commonly found in literature as the reparametrization trick [15]. This technique reparemetrizes the arbitrary Gaussian distribution $q(z|x)$ by allowing samples to be generated using learned mean and standard deviance parameters and a noise sample from a normal distribution (Equation 35).

$$z = \mu + \sigma \odot \epsilon, \epsilon \sim \mathcal{N}(0, I) \tag{35}$$

The normal distribution will be an auxiliary input for the network and parameters μ, σ can be learned and are differentiable. As the generator also models a distribution $p(x|z)$, the same reparametrization can be applied here.



(a) Full architecture of the variational autoencoder



(b) Genereting a sample using only the generator of the VAE

Figure 11a shows the entire architecture of a variational autoencoder. The loss of the model for each example is a combined loss for both encoder and decoder. Commonly, the loss is calculated using the L2-norm of reconstruction error. VAEs, like any feed-forward neural network can be trained with batches, but given a complex distribution of e.g. images, require a large number of training samples and many iterations over the dataset. Figure 11b shows the generation process of the VAE. As the goal is to model a latent distribution of the input data, to generate a sample, one can choose an arbitrary point within the normally distributed feature space and feed this sample through the decoder network to recieve an output distribution which can either be sampled from or directly be mapped to data. VAEs can model distributions of a chosen dimensionality which allows to model Images using only two dimensions where both dimensions have direct impact on the outcome of the generator [5].

5.2 Applying a VAE

To show off what a VAE is able to generate, we use MNIST, scaled to real valued $(0, 1)$. Therefore we do no longer model binary but now real valued images. We skip modeling the output of the decoder as a distribution and directly use output values as pixel values. Both encoder and decoder network have one hidden layer with 512 neurons with ReLU [18] activation function. The encoder network has a hidden layer with two neurons to output a distribution with two

dimensions. The decoder respectively uses a 784 neuron layer to reconstruct the original MNIST input image. We input images in batches of 128 and train the network for 50 epochs on the full 50000 MNIST samples. We then construct a grid of samples based on the latent space which can be seen in Figure 12

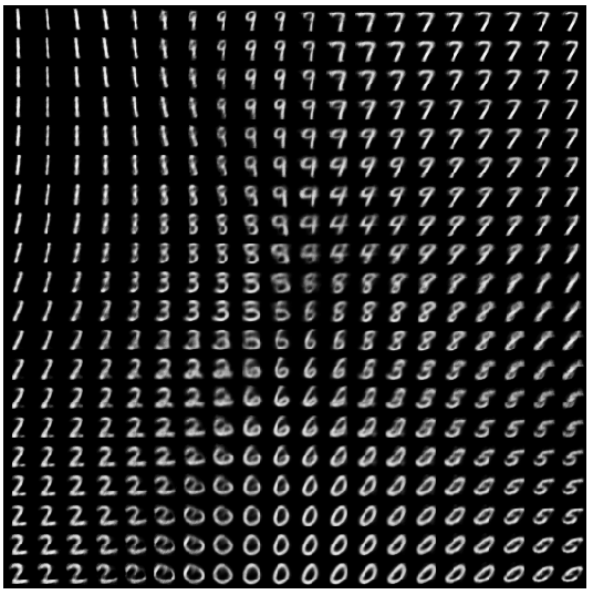


Figure 12: VAE on MNIST, latent distribution values in $-2, 2$

Using loss metrics such as L2, Output of VAEs are blurry, particularly in areas where high variance of $q(x|z)$ is observed. This problem can be attributed to the limitation of modeling any image using Gaussian distributions. However, real-world images often are not based only on a mixture of Gaussian distributions and the VAE fails to accurately model some parts of the image [33]. Images are not the only type of data generative models can generate. Yet, they are complex and it is easily recognizable by a human if an image has been accurately generated or not. Moving away from explicitly modeling distributions, generative adversarial networks have shown great success in generating sharp high resolution images [3].

6 Generative Adversarial Network

Generative Adversarial Networks (GAN) build on the movement away from explicitly modeling the distribution $p(x)$ using a maximum likelihood approach. Instead, the entire learning is done via a two player zero-sum minimax game. The two players consists of two function approximators, multi-layer neural networks, where on player generates images from random noise and the other player tries to decide whether an input image has be generated by the other or is a sample from an input dataset. Therefore, the generator G learns a sampling process which produces images similar to the dataset ($p(x)$) in order to trick the discriminator D [10]. The architecture of a basic GAN is depicted in Figure 13 where the random noise z can be of arbitrary length but ideally reflects the complexity of the input dataset. Samples generated by the generator from noise are initially very far from the real dataset, but approach real images over many training examples and epochs.

6.1 Training a GAN

To train a GAN, both G_{θ_g} and D_{θ_d} are parameterized by using feed-forward neural networks with weights. The minimax game is then formulated by including discriminator and generator with the discriminator trying to maximize the equation and the discriminator trying to minimize the maximization of G (Equation 36) [10].

$$\min_{\theta_g} \max_{\theta_d} \Big[\mathbb{E}_{x \sim p_{data}}[log(D_{\theta_d}(x))] + \mathbb{E}_{z \sim p(z)}[log(1 - D_{\theta_d}(G_{\theta_g}(z)))] \Big] \tag{36}$$

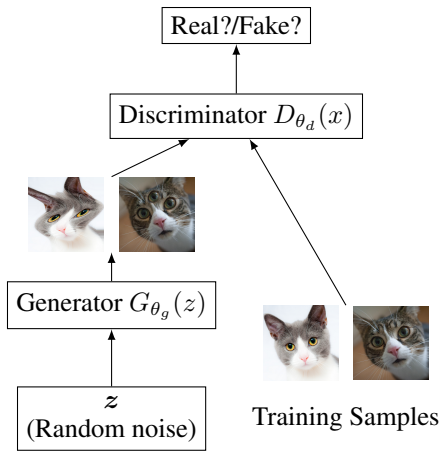


Figure 13: GAN architecture

In this game, D will maximize the entire term, as it is included in both expectations such that the discriminator maximization can be written as in Equation 37. Using the information from the equation, discriminator weights θ_d can be adjusted using stochastic gradient ascent.

$$\max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}}[log(D_{\theta_d}(x))] + \mathbb{E}_{z \sim p(z)}[log(1 - D_{\theta_d}(G_{\theta_g}(z)))] \right] \tag{37}$$

The first expectation in Equation 37 represents how good the discriminator can recognize real data correctly and the second expectation how well generated data can be recognized as such. G will be trained by minimizing the second part of the equation using gradient descent. Considering, the shape of the log curve $log(1 - x)$ in regard to its gradient, gradient descent might be sub-optimal in this case. If the discriminator assigns a probability of zero to the generators sample, the gradient should be the highest, not the lowest. Figure 14 shows $log(1 - x)$ and $-log(x)$, but the same

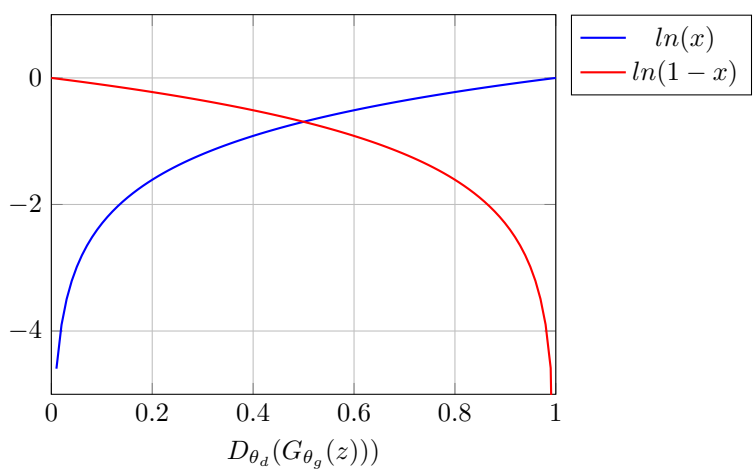


Figure 14: Loss compared to discriminator assigned probability

positive effects of using $-log(x)$ hold for using $log(x)$. By optimizing the generator using $log(x)$, the issue of low gradients when the discriminator is correct is alleviated. Therefore, in practice Equation 38 is used to optimize the generator.

$$\max_{\theta_g} \left[\mathbb{E}_{z \sim p(z)}[log(D_{\theta_d}(G_{\theta_g}(z)))] \right] \tag{38}$$

Having set up the game, the GAN training algorithm is simple:

1. m samples $\{\boldsymbol{z}^{(1)}, ..., \boldsymbol{z}^{(m)}\}$ are drawn from a multi-dimensional noise prior $p(z)$. The same number of examples $\{\boldsymbol{x}^{(1)}, ..., \boldsymbol{x}^{(m)}\}$ are then drawn from the training dataset.

2. Using the samples, the discriminator D_{θ_d} is trained given a label if the samples is generated or drawn from real data. For gradient calculation, the mean over the entire mini-batch is taken.
3. Step one and step two are repeated for k iterations
4. From the same noise distribution, new samples $\{\boldsymbol{z}^{(1)}, ..., \boldsymbol{z}^{(m)}\}$ are drawn.
5. Generator G_{θ_g} is trained using the new sample set with training labels coming from $D_{\theta_d}(D_{\theta_d}(z))$
6. Repeat the entire process for the desired number of training iterations.

Training GANs requires large amounts of data, with the most recent prominent examples of Brock et al. (2018) [3] using 292.000.000 images training over 200.000 iterations before stopping. Put into context of computational requirements, they trained their network for up to 48h on 512 tensor processing units. Yet, after successful training, the generator can then be used separately to generate new data samples from the noise distribution just as it generated data during training. In practice successful training is one of the main problems of GANs.

6.2 Mode Collapse

One very common issue in training GANs is mode collapse, where the generator learns to generate data with low sample variety. Given a large range of inputs the generator always outputs the same values. During training, the generator learns a specific mode which is classified as real. Over time, the discriminator will learn that the generator always produces the same mode and will randomly guess for similar samples and instead discern every other input data as real. Instead of learning to generate data for this mode and any other mode, the generator moves to the next mode which was previously classified as always real by the discriminator. This process then repeats until training is stopped [8]. To improve sample variety one can evaluate the similarity of batches of generated images and adjust if images are too similar [28].

6.3 Applying a GAN

As an example, we again use the MNIST dataset to train a GAN. In practice, the GAN generator loss has frequent spikes where the loss increases strongly again and mode collapse is a common issue as well. A number of improvements have been proposed to avoid such issues since the publication of the original GAN paper, however our implementation will try to stay as close to our explanation as possible. For the discriminator network, we use two fully connected layers with 512 and 265 neurons respectively with the final activation function being sigmoid to decide on a $0, 1$ outcome. The generator is a three layer network with 256,512 and 1024 neurons with batch normalization [14] to speed up training and allow the generator to keep up with the discriminator. Both networks use leaky ReLU [18] as activation functions in intermediate layers. The MNIST dataset is also normalized to zero mean and unit variance. Therefore the output activation function of the generator will be the hyperbolic tangent. We train both generator and discriminator for one step for every batch with batch size 32. The entire network is trained over 30000 iterations. Lastly the noise vector for the generator input is size 100. Results are shown in Figure where 25 random GAN samples are displayed.

7 Discussion

Generating new data by learning from a set of input data is possible in particular for data with low complexity and few input dimensions. Given the rise of deep learning as a means to handle more complexity, even sound and images can be generated with high fidelity. Brock et al. (2018) [3] demonstrate how far current architectures have come, yet they also show that the required computational power is far beyond anything a computer at home can do. However they also demonstrate that MCMC based approaches are falling out of favor because no faster and more accurate approaches have been found to handle large amounts of data. Furthermore instead of using a specific maximum likelihood estimation, optimization is done via stochastic gradient descent. Given that there is proof that for a large spectrum of functions SGD converges [21] it is likely that the introduced group of Boltzmann machines will see even less use in the future. It should be briefly addressed, that a type of generative model has not been explained in this work. All generative models using variants of regression to predict e.g. pixels have been omitted in particular because the most prominent proposed algorithm PixelRNN [22] seems to not have gained much traction in particular compared to GAN and VAE. The number of papers submitted with different GAN architectures has been growing steadily on the other hand [11]. One important issue with GANs compared to the other introduced algorithms, reverse inference, has already been solved in a recent work as well. Given that GANs are very simple to implement and are already very popular, it is likely that they will stay the state-of-the-art approach to deep generative models for some time.

8 Code

Python code for all written examples in this work can be found on Github: `https://github.com/eliasbaumann/Intro-deep-generative-models`.

References

[1] David H Ackley, Geoffrey E Hinton, and Terrence J Sejnowski. A learning algorithm for boltzmann machines. *Cognitive science*, 9(1):147–169, 1985.

[2] Mariette Awad and Rahul Khanna. *Efficient learning machines: theories, concepts, and applications for engineers and system designers*, page 135. Apress, 2015.

[3] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale gan training for high fidelity natural image synthesis. *arXiv preprint arXiv:1809.11096*, 2018.

[4] Edward Choi, Siddharth Biswal, Bradley Malin, Jon Duke, Walter F Stewart, and Jimeng Sun. Generating multi-label discrete patient records using generative adversarial networks. *arXiv preprint arXiv:1703.06490*, 2017.

[5] Carl Doersch. Tutorial on variational autoencoders. *arXiv preprint arXiv:1606.05908*, 2016.

[6] Asja Fischer and Christian Igel. An introduction to restricted boltzmann machines. In *Iberoamerican Congress on Pattern Recognition*, pages 14–36. Springer, 2012.

[7] Ove Frank and David Strauss. Markov graphs. *Journal of the american Statistical association*, 81(395):832–842, 1986.

[8] Ian Goodfellow. Nips 2016 tutorial: Generative adversarial networks. *arXiv preprint arXiv:1701.00160*, 2016.

[9] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*, pages 651–716. MIT Press, 2016. `http://www.deeplearningbook.org`.

[10] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[11] Avinash Hindupur. The gan zoo. `https://github.com/hindupuravinash/the-gan-zoo`, 2018.

[12] Geoffrey E Hinton. Training products of experts by minimizing contrastive divergence. *Neural computation*, 14(8):1771–1800, 2002.

[13] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

[14] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[15] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[16] Solomon Kullback and Richard A Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.

[17] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[18] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, page 3, 2013.

[19] Abdel-rahman Mohamed and Geoffrey Hinton. Phone recognition using restricted boltzmann machines. In *Acoustics Speech and Signal Processing (ICASSP), 2010 IEEE International Conference on*, pages 4354–4357. IEEE, 2010.

[20] Kevin P Murphy. *Machine Learning: A Probabilistic Perspective*, pages 995–998. MIT Press, 2012.

[21] Lam M Nguyen, Phuong Ha Nguyen, Peter Richtárik, Katya Scheinberg, Martin Takáč, and Marten van Dijk. New convergence aspects of stochastic gradient algorithms. *arXiv preprint arXiv:1811.12403*, 2018.

[22] Aaron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks. *arXiv preprint arXiv:1601.06759*, 2016.

[23] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[24] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. *arXiv preprint arXiv:1401.4082*, 2014.

[25] Ruslan Salakhutdinov. Learning deep generative models. *Annual Review of Statistics and Its Application*, 2:361–385, 2015.

[26] Ruslan Salakhutdinov and Geoffrey Hinton. Deep boltzmann machines. In *Artificial Intelligence and Statistics*, pages 448–455, 2009.

[27] Ruslan Salakhutdinov and Hugo Larochelle. Efficient learning of deep boltzmann machines. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 693–700, 2010.

[28] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. In *Advances in Neural Information Processing Systems*, pages 2234–2242, 2016.

[29] Lawrence K Saul, Tommi Jaakkola, and Michael I Jordan. Mean field theory for sigmoid belief networks. *Journal of artificial intelligence research*, 4:61–76, 1996.

[30] Tijmen Tieleman. Training restricted boltzmann machines using approximations to the likelihood gradient. In *Proceedings of the 25th international conference on Machine learning*, pages 1064–1071. ACM, 2008.

[31] Max Welling, Michal Rosen-Zvi, and Geoffrey E Hinton. Exponential family harmoniums with an application to information retrieval. In *Advances in neural information processing systems*, pages 1481–1488, 2005.

[32] Alan L Yuille. The convergence of contrastive divergences. In *Advances in neural information processing systems*, pages 1593–1600, 2005.

[33] Shengjia Zhao, Jiaming Song, and Stefano Ermon. Towards deeper understanding of variational autoencoding models. *arXiv preprint arXiv:1702.08658*, 2017.