RWTH Aachen University
Chair of Computer Science 6
Prof. Dr.-Ing. Hermann Ney

Selected Topics in Human Language Technology and Pattern Recognition WS 16/17

Deep Directed Generative Models

André Merboldt

346703

March 3, 2017

Supervisor: Tobias Menne

Contents

List of Figures

Abstract

Deep directed generative models leverage the possibilities of deep learning to learn rich, hierarchical structures from datasets in an unsupervised context. These models can be used to understand high-dimensional, real-world data and transform them into compact representations. Similar to how humans learn to generalize from sensory input, we discuss several models which can learn compact, internal representational structures. After discussing generative modeling in general and its applications, we introduce the sigmoid belief network which is a simple generative architecture. We then focus on two more recently proposed architectures: variational autoencoders, and generative adversarial networks. We discuss the key idea of both models and their respective architecture and training procedure, including variational inference and adversarial training for the variational autoencoder and generative adversarial network respectively. Additionally to the originally proposed models, we summarize several extensions and present experimental results for the variational autoencoder.

1 Introduction

Intuitively, generative modeling is based on the idea that in order to produce data similar to samples from a dataset it helps to understand the data first [9, p. 720]. In contrast to discriminative models which model the conditional distribution $p(y|x)$ of the labels y given observations x, generative algorithms model the joint distribution $p(x, y)$ over both labels and observations to describe how the data was generated [13]. As there are only observations given, the generative task seems more difficult than classification where both input and desired output are given during training. [9, p. 695]

Due to the nature of generative models, they are unsupervised learning algorithms which mean that they try to learn the structure of samples from an unlabeled dataset[9, p. 105-106][8]. Generative models in machine learning using probabilistic models can be mostly separated into directed and undirected models [9, p. 77]. This separation indicates how the model can be mapped to a graph with random variables as vertices and interactions between random variables as edges. Directed models induce a directed network graph while undirected models induce a network graph with undirected edges.

In directed models, we assume that the observations x are stochastically dependent on the underlying factors, also called latent variables z. This relationship can be expressed as the graphical model shown in Figure 1. Note that the latent variables can be structured hierarchically in order to express different levels of granularity in the features of the underlying factors and representations [9, p. 557].

1.1 Overview

Traditional models include hidden Markov models used for example in speech recognition [26], which are generative models [31, p. 878] and Gaussian mixture models [9, p. 190][34].

Undirected generative models include various extensions of the Boltzmann machine [2] such as the restricted Boltzmann machine (RBM), first described as Harmonium [35] and the deep Boltzmann machine [32][9, Chapter 20.4].

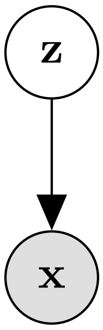


Figure 1: Directed Graphical Model

The observed data x is stochastically dependent on the latent variables z.
Based on [3, Chapter 8].

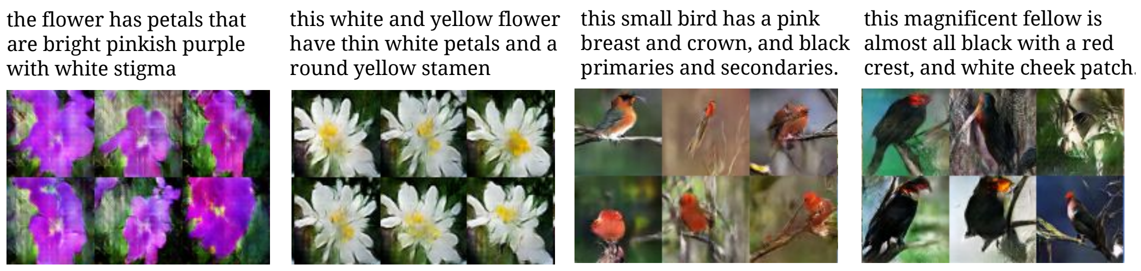

the flower has petals that are bright pinkish purple with white stigma
this white and yellow flower have thin white petals and a round yellow stamen
this small bird has a pink breast and crown, and black primaries and secondaries.
this magnificent fellow is almost all black with a red crest, and white cheek patch.

Figure 2: Generative Text to Image Synthesis [28]

The top row shows the provided captions and in the bottom row the corresponding synthesized images.

For directed models, the sigmoid belief network (SBN) is an early model described in 1992 [23] which is presented in Section 2. SBN has also been extended with a restricted Boltzmann machine for the first two layers resulting in deep belief nets (DBN) [13], which have been attributed to spark the beginning of current deep learning [9, p. 662]. DBN are partially directed models, as they contain both directed and undirected connections [9, p. 664].

1.2 Applications

Generative models learn the joint distribution $p(x, y)$ and due to this fact they can be used to perform classification as well by expand $p(x, y)$ to $p(y|x)$ as indicated by Equation 1.

$$p(y|x) = \frac{p(x, y)}{p(x)} \tag{1}$$

One obvious application of generative models is to use the model to sample more data similar to samples from a given training dataset.
In computer vision tasks, generative models have been largely outperformed by much simpler models until at least 2011 [38]. In 2015 however, convolutional networks have been shown to surpass those models in image synthesis [7]. Other tasks involving image modeling include performing super-resolution [37], constructing 3D models from 2D images [41],

predicting the next video frames [39], and synthesizing new images from captions as shown in Figure 2.
In the area of reinforcement learning, generative models offer a way to perform action space exploration [15]. Generative adversarial networks discussed in Section 4 have been shown to be used in imitation learning, where the generative model is used to imitate previously seen interactions [14].
SampleRNN [22], which is a recurrent neural network combined with an auto-regressive network have been shown to produce good generated audio samples.

Recently, generative models have also shown to improve semi-supervised learning performance significantly [17]. Semi-supervised learning algorithms use only small amount of labeled data, which is usually difficult to obtain and large amounts of unlabeled data, which is easier to obtain in most cases.

2 Sigmoid Belief Networks

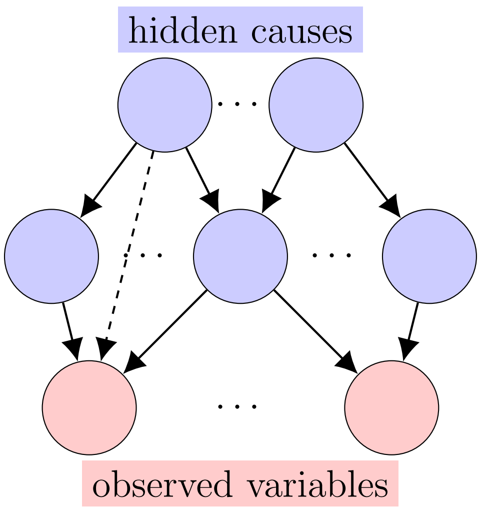


Figure 3: Conceptual architecture of belief nets

Bayesian networks, also called belief nets are an acyclic directed graph where blue nodes indicate hidden variables and red nodes represent observed variables. Even though most belief nets are divided into many layers in the context of deep learning [9, Chapter 20.10.1], the original definition does not specify that and thus indicates the dashed arrow the possibility of inter-layer dependencies.

Sigmoid Belief Networks (SBN) were described in 1992 by R.M. Neal [23] representing a specific type of bayesian networks [25] shown in Figure 3 In the case of SBNs however, all states are binary and the activation function is the sigmoid function. They are one of the earliest neural networks used for generative modeling, predating all other presented directed models in this article. Each state s_i is a random variable in the bayesian framework which means it can be observed, hidden or unknown representing different states of knowledge. As such it can also be interpreted with a probability distribution.

$$p(s_i) = \sigma\left(\sum_{j<i} W_{j,i}s_j + b_i\right) \tag{2}$$

where

$\sigma(x) = \frac{1}{1+e^{-x}}$ (sigmoid function)
$W_{j,i}$ = connection weights from state s_j to s_i

To draw samples from this network, it is possible to evaluate Equation 2 for each visible node. However, prior to that the SBN first needs to learn how to generate good samples which includes the *learning* and the *inference* problem [9, p. 695].
Learning means in this context to adjust the weights and biases of the network to be more likely to generate data similar to samples from the given training dataset. Meanwhile, statistical inference is tasked to infer the states of the hidden nodes based on the states in the visible layer.
Due to each state being statistically dependent on all ancestors, inference means to evaluate all possible configurations of every previous node. Especially in deep networks, this kind of computation is infeasible as it requires computing exponentially many configurations.

3 Variational Autoencoders

Variational autoencoders (VAE) have been developed concurrently in 2014 by Kingma et al.[18] and Rezende et al.[29]. The general idea of VAEs is to have an encoder-decoder architecture where input x gets mapped to a specific distribution defined in the latent space z from which we sample in order to reconstruct the input.

The probabilistic encoder generally implemented as feed-forward network receives input from the dataset and outputs parameters of a distribution. For example the parameters for a gaussian distribution with diagonal covariance, in which case the network emits the mean μ and σ used to define the gaussian distribution $\mathcal{N}(\mu, \sigma \times I)$, where I is the identity matrix.

Using gradient-based optimization techniques enables the encoder network to be trained to output distributions similar to the true posterior $p(z|x)$. In addition to this inference network, there is a generator network which transforms points z in the latent space back to the data space while trying to minimize the reconstruction error.

One useful aspect of this approach is the possibility to train both networks using error backpropagation due to the use of neural networks as function approximators.

Such function approximator will emit the variational parameters ϕ of the probability encoder distribution $q_\phi(z|x)$. Equally for the probabilistic decoder which constructs a distribution $p_\theta(x|z)$ with the parameters θ. This architecture is shown in Figure 4 with a one-dimensional Gaussian prior.

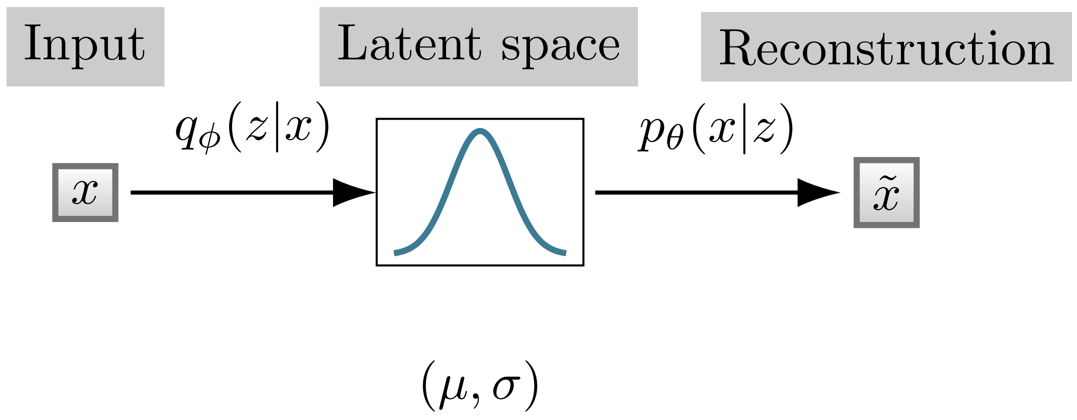


Figure 4: simplified VAE architecture using a gaussian prior on z

The key insight on how the variational autoencoder is able to optimize $q_\phi(z|x)$ to approach the true data posterior distribution $p(z|x)$ is in defining a variational lower bound which is used to push the approximated $q_\phi(z|x)$ towards $p(z|x)$. We will discuss this lower bound $\mathcal{L}$ and its derivation in detail in the following sections.

3.1 Approximated Inference

A problem which often arises in directed generative models is the *inference problem* [9, Chapter 16.6]. As discussed in 1, we assume that the observations are stochastically dependent on some latent variables. Inference is called the process of coming up with values of z which can produce the observations, which formally means evaluating $p(z|x)$ where z are the latent variables and x observations.

$$p(z|x) = \frac{p(x|z)p(z)}{p(x)} \tag{3}$$

where

$p(z\|x)$	= posterior distribution
$p(x\|z)$	= likelihood of x given z
$p(z)$	= prior distribution of z

Using Bayes's rule we can expand the posterior distribution $p(z|x)$ as shown in Equation 3. Note that to evaluate the denominator $p(x)$, all hidden variables need to be marginalized: $p(x) = \int p(x,z)dz$.
Due to the possibly large number of hidden variables, this integral can be high-dimensional and therefore difficult to calculate. This integral can be difficult to compute and needs to be approximated. However, for a large number of hidden variables, this integral is intractable to compute exactly and therefore needs to be approximated [3, p. 461-462]. In order to approximate those intractable posterior distributions, we discuss two commonly used techniques: Markov chain Monte Carlo and variational inference.

Markov chain Monte Carlo MCMC methods are a class of techniques which can be used to sample from a large class of distribution where direct sampling is not applicable [3, Chapter 11.2]. For the variational autoencoder, where we want to approximate the posterior distribution $p(z|x)$ a method called *Gibbs sampling* [3, Chapter 11.3] can be used to obtain samples from that conditional distribution by constructing a Markov chain with the desired equilibrium distribution. One advantage of MCMC methods is that the resulting distribution is proven to be exactly the desired distribution [3, Chapter 11.3]. However, due to the problems of mixing between mixing modes [9, Chapter 17.5] and slow convergence rate[9, p. 603] these methods are impractical for usage in the inner loop of optimization techniques.

Variational Inference Due to the possibly slow convergence of MCMC methods, variational inference (VI) [9, Chapter 19.4][3, Chapter 10.1] is used in the variational autoencoder [18]. VI works by picking a specific distribution restricted to a family of distributions which matches the posterior most closely. In the case of the VAE, we use neural networks to select the parameters of the distributions to turn the inference problem into an optimization problem, allowing for gradient-based optimization. Due to the restriction of VI to a specific probability distribution class and neural networks as non-convex function approximators, it is not guaranteed to converge. Conclusively, even though MCMC offers approximation with guaranteed convergence [3, Chapter 11.2], in practice it is often slow to converge while VI offers reasonable good approximation at greater speed [18].

3.1.1 Kullback-Leibler Divergence

In order to approximate one distribution Q to another distribution P, we need to have a measurement of how close Q is to P. One commonly used measurement is called Kullback-Leibler (KL) divergence [20] which enables the variational inference methods to choose appropriate distributions. The KL divergence from Q to P is defined in Equation 4.

$$\mathcal{D}_{\mathrm{KL}}(P||Q) = \mathbb{E}_{x \sim P(x)}\left[log \frac{P(x)}{Q(x)}\right] \tag{4}$$

Note that divergence does not obey the triangle inequality and is also not symmetrical, therefore it does not qualify as a metric. $\mathcal{D}_{\mathrm{KL}}(P||Q)$ can nevertheless be understood as a measure of the difference between P and Q and is as such used in the variational inference.

3.2 Objective Function

Following the thoughts earlier, we wish to derivate an objective function which can be used to perform gradient-based optimization in the model.
We construct the approximate posterior $q_\phi(z|x)$ which we want to approach the true data posterior $p(z|x)$. When using the KL divergence as measurement for difference between two distributions the goal for which we want to optimize becomes the following.

$$q^*_\phi(z|x) = \arg\min_\phi \mathcal{D}_{\mathrm{KL}}\big(q_\phi(z|x)||p(z|x)\big) \tag{5}$$

However, we are not able to optimize Equation 5 directly, as the intractable evidence $p(x)$ appears during the expansion of the posterior $p(z|x)$ to $\frac{p(x|z)p(z)}{p(x)}$.
The denominator $p(x)$ can be computed by marginalizing out the hidden variables z:

$$p(x) = \int p(x,z)p(z)dz \tag{6}$$

Computing the integral in Equation 6 requires the evaluation of every possible configuration over the hidden variables z and is therefore intractable.

Instead of directly optimizing for the true posterior, we construct an evidence lower bound (ELBO) $\mathcal{L}$ which we can maximize which is equivalent to minimizing the KL divergence between $q_\phi(z|x)$ and $p(z|x)$ without explicity evaluating the evidence.

Derivation of the evidence lower bound Based on [9, p. 698].

$$
\begin{aligned}
&\log p(x) \\
&\quad = \int_z q_\phi(z|x) \log p(x) dz \\
&\quad = \int_z q_\phi(z|x) \log \frac{p_\theta(x,z)}{p_\theta(z|x)} dz && \text{(Bayes' rule)} \\
&\quad = \int_z q_\phi(z|x) \log \left(\frac{p_\theta(z,x)}{q_\phi(z|x)} \frac{q_\phi(z|x)}{p_\theta(z|x)} \right) dz && \text{(Chain rule)} \\
&\quad = \int_z q_\phi(z|x) \left(\log \frac{p_\theta(z,x)}{q_\phi(z|x)} + \log \frac{q_\phi(z|x)}{p_\theta(z|x)} \right) dz \\
&\quad = \int_z q_\phi(z|x) \log \frac{p_\theta(z,x)}{q_\phi(z|x)} dz + \int_z q_\phi(z|x) \log \frac{q_\phi(z|x)}{p_\theta(z|x)} dz \\
&\quad = \underbrace{\int_z q_\phi(z|x) \log \frac{p_\theta(z,x)}{q_\phi(z|x)} dz}_{=\mathcal{L}} + \underbrace{\mathcal{D}_{\mathrm{KL}}\big(q_\phi(z|x) \| p_\theta(z|x)\big) dz}_{\geq 0} \\
&\quad \geq \mathcal{L}
\end{aligned}
$$

$\mathcal{L}$ is called the evidence lower bound with respect to $\log p(x)$. We further rewrite $\mathcal{L}$ in order decompose the term into two separate terms, the reconstruction error $\mathcal{L}_{rec}$ and the regularizer $\mathcal{L}_{reg}$.

$$
\begin{aligned}
\mathcal{L} &= \int q_\phi(z|x) \log \frac{p_\theta(z,x)}{q_\phi(z|x)} dz \\
&= \int q_\phi(z|x) \log \frac{p(z) p_\theta(x|z)}{q_\phi(z|x)} dz \qquad \text{(we assume that x is conditionally dependent on z)} \\
&= \int q_\phi(z|x) \left(\log \frac{p(z)}{q_\phi(z|x)} + \log p_\theta(x|z) \right) dz \\
&= -\int q_\phi(z|x) \log \frac{q_\phi(z|x)}{p(z)} dz + \int q_\phi(z|x) \log p_\theta(x|z) dz \\
&= -\mathcal{D}_{\mathrm{KL}}\big(q_\phi(z|x) \| p(z)\big) + \int q_\phi(z|x) \log p_\theta(x|z) dz \\
&= \underbrace{-\mathcal{D}_{\mathrm{KL}}\big(q_\phi(z|x) \| p(z)\big)}_{\mathcal{L}_{reg}} + \underbrace{\mathbb{E}_{z \sim q_\phi(z|x)} \big[\log p_\theta(x|z) \big]}_{\mathcal{L}_{rec}} \\
&= \mathcal{L}_{reg} + \mathcal{L}_{rec}
\end{aligned}
$$

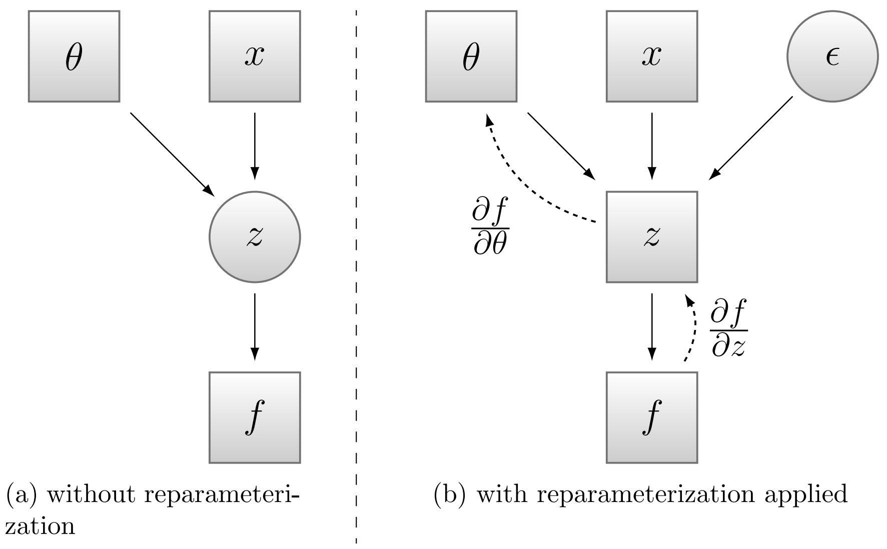

θ
x
z
f
(a) without reparameterization
θ
x
ϵ
z
f
$\frac{\partial f}{\partial \theta}$
$\frac{\partial f}{\partial z}$
(b) with reparameterization applied

Figure 5: Reparameterization Trick

Function $f_\theta(x)$ with inputs x and (variational) parameters θ as computational graph. Left without and on the right with the reparameterization trick applied.
Solid arrows show the forward pass and dashed arrows the error backpropagation using gradients.
Rectangular nodes represent deterministic functions while circles indicate stochastic variables.

3.2.1 Reparameterization Trick

To be able to apply the backpropagation[30] algorithm to the loss function of the VAE without any injected noise it has to be both differentiable and deterministic. However, as z is sampled from the constructed probability distribution $q_\phi(z|x)$ it is stochastic. This poses a problem because the gradient can not be computed accurately due to the added noise.
In order to circumvent this restriction, the reparameterization trick [18][29][9, chapter 20.9] can be applied. Instead of drawing z directly from $q_\phi(z|x)$, we first sample an auxiliary variable ϵ from the isotropic gaussian $\mathcal{N}(0, I)$. Then we apply Equation 7 to transform ϵ into z.

$$z = \mu(x) + \Sigma^{1/2}(x) * \epsilon \tag{7}$$

Note that this transformation is entirely deterministic and as such allows for backpropagation through the transformation into the variational parameters ϕ.

Figure 5 expresses the reparameterization trick applied to an arbitrary function f with the parameters θ.
Using this trick, the whole VAE model can be trained using gradient-based optimization techniques.

3.3 Learning

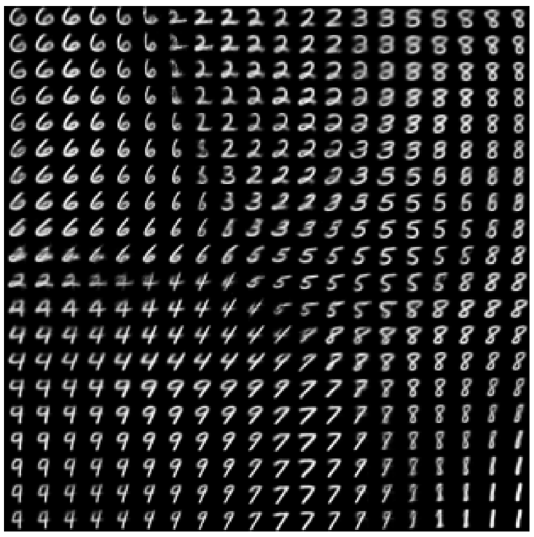
The probabilistic encoder $q_\phi(z|x)$ and decoder $p_\theta(x|z)$ are commonly implemented using deep neural networks. To allow analytical computation of the ELBO, the prior on the latent space z is often choosen to be the isotropic gaussian $p(z) = \mathcal{N}(0, 1)$. The learning procedure of a variational autoencoder is shown in Algorithm 1 which is a simplified version of the algorithm in the original paper [18].

Algorithm 1 Learning in the VAE model

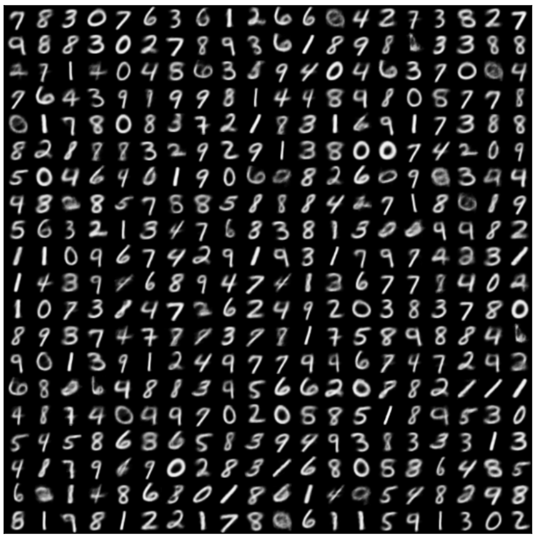
```
1: while has not reached convergence do
2:     $x \leftarrow$ sample from $p_{data}(x)$
3:     $\epsilon \leftarrow$ Random samples from noise prior $p(\epsilon)$
4:     $g \leftarrow$ Compute gradients $\nabla_{\theta,\phi}\mathcal{L}(\theta, \phi; x, \epsilon)$
5:     $\theta, \phi \leftarrow$ Update according to gradients $g$
6: end while
```

3.4 Experiments

In order to evaluate the inference performance and explore the learned latent space, the VAE has been implemented and trained using TensorFlow [1] on the MNIST dataset implementing a fully-connected network.Both experiments as shown in Figure 6 have been performed with two hidden layers of size 500. Rectified linear units [12] were used as activation functions in this network.



(a) Learned manifold



(b) Randomly generated images by sampling

Figure 6: Manifold Learning in the VAE

(a) Evaluation of a variational autoencoder with 2 hidden layers with each 500 nodes and a two-dimensional latent space. The shown manifold has been plotted using samples uniformly samples in the range $[-3, -3]^2$ and then reconstructed using the decoder network $p_\theta(x|z)$. (b) generative performance qualitatively evaluated using samples in the data space by random sampling $z \sim p(z)$ and decoding z using the decoder network. For this experiment a 50-dimensional latent space has been used.

3.5 Extensions

3.5.1 Conditional VAE

Instead of using VAEs in a completely unsupervised manner on an unlabeled dataset, it is also possible to learn from datasets where a small subset of data points has corresponding labels while most data points are still unlabeled. Techniques learning from such datasets are called semi-supervised learning and can improve performance significantly [17][36].
To adjust the VAE model to the semi-supervised case, we have to add a random variable y corresponding to the label if available.

First, instead of using samples from the dataset alone we are now also conditioning on y, formally $x \sim p_\theta(x|y,z)$ instead of $x \sim p_\theta(x|z)$.
Then the probabilistic encoder and decoder can be written as $q_\theta(z|x,y)$ and $p_\phi(x|y,z)$ respectively.
These modifications allows us to reformulate the objective function $\mathcal{L}$ from Equation 5 in equations 8 and 9.

For datapoints x where labels y are available.

$$\log p_\theta(x) \geq \mathbb{E}_{q_\phi(z|x,y)}\Bigg[\log p_\theta(x|y,z) + \log p_\theta(y) + \log p(z) - \log q_\phi(z|x,y)\Bigg] \tag{8}$$

And for the unsupervised case where the label y corresponds to an unknown entity, we treat y as an additional latent variable.

$$\log p_\theta(x) \geq \mathbb{E}_{q_\phi(y,z|x)}\Bigg[\log p_\theta(x|y,z) + \log p_\theta(y) + \log p(z) - \log q_\phi(y,z|x)\Bigg] \tag{9}$$

The final objective function forthe whole dataset is the summation between right-hand terms of equations 8 for the labeled subset and 9 for the unlabelled subset.

3.5.2 Deep Latent Gaussian Model

Up until now the presented variational autoencoder framework has only considered one stochastic latent layer. However in the original paper by Rezende et al. [29] the authors have shown a variational autoencoder using multiple stochastic layers with gaussian priors in each layer. They called this model the deep latent gaussian model (DLGM) which can be understood as a generalization of the variational autoencoder. Note that the deterministic non-linear transformations between each layer still can be represented using deep neural networks.
Using multiple stochastic layers enables the model to learn more hierarchical structured dependencies from the dataset.

3.5.3 Importance Weighted Autoencoder

The Importance Weighted Autoencoder[4] (IWAE) extends on the VAE framework using k-sampled importance weighting of the log-likelihood. Using importance weighting can be informally interpreted as putting more weight on data with high likelihood.

Improving on the theoretical results in the VAE framework, using IWAE with more samples will tighten the lower variational bound and therefore improve the overall performance of the model.

$$\log p(x) \geq \mathcal{L}_{k+1}(x) \geq \mathcal{L}_k(x)$$

Additionally, if $\frac{p(h,x)}{q(h|x)}$ is bounded, then the model is guaranteed to reach the optimum as k approaches infinity in equation 10

$$\lim_{k\to\infty} \mathcal{L}_k(x) = \log p(x) \tag{10}$$

However, using more samples will result in higher computational cost and thus should be carefully evaluated.

3.5.4 Deep Recurrent Attention Writer

The Deep Recurrent Attention Writer[11] (DRAW) is an advanced extension of the variational autoencoder. It uses recurrent neural networks as encoder as well as decoder and includes a selective attention mechanism. These advancements allow the model to observe an image through a series of *glimpses* and draw to an output region determined by the attention mechanism. Without going into much detail, this allows the model to sequentially draw images in a way which resembles much of human-like drawing or sketching.

4 Generative Adversarial Networks

Generative Adversarial Networks (GAN) is a framework proposed in 2014 by Goodfellow et al. [10] based on a game-theoretic approach to generative modeling [9, Chapter 20.10.4].

GANs circumvent the issues arising around the inference problem by playing a two-player game. Both players play non-cooperatively against each other, which means they only try to maximize their respective success. One player is called the generator G which tries to produce authentic looking data samples while the other player, the discrimininator D has to decide whether a received sample is generated or actually real data. It is important to note that the generator gets to know whether the discriminator accepted the generated output as real or rejected it as fake.

As each party plays to maximize its own success, the discriminator will try to maximize the number of samples which got discriminated correctly. On the other side, the generator tries to produce samples aiming to fool the discriminator.

As the game progresses, both parties will get better at their respective objective and the generated samples will look more and more like the training examples which is the ultimative goal is generative modeling.

Solution The game-theoretic solution to the emerging situation where neither G nor D can improve its own success with respect to the outcome of the game gets called Nash equilibrium[24]. In the case of GANs, this situation arises when the generator produces samples which get rejected with probability $p = 0.5$ by the discriminator. In the case of such equilibrium, the generator produces good enough samples to fool the generator in about half cases.

Another formulation of the same game is to define the zero-sum game between the generative model and its adversary, the discriminator [9, Chapter 20.10.4]. The payoff of the discriminator can be described as the value function $V(G, D)$ while the payoff of the generator is the negative payoff of D: $-V(G, D)$.
This formulation allows us to define the reached nash equilibrium in equation 11.

$$G^* = \arg\min_G \max_D V(G, D) \tag{11}$$

4.1 Architecture

To formalize the setting in which the game between G and D takes place, we define the true, however unknown data distribution $p_{data}(x)$ from which samples are available as dataset $X = \{x^{(1)}, x^{(2)}, \ldots, x^{(N)}\}$.

In the case of generative modeling, the goal is to approximate $p_{data}(x)$ with $p_{model}(x)$. GANs uses neural network as function approximator for G as well as D. The generator network G produces samples from $p_{model}(x)$ by transforming samples z from random noise as $x = G(z)$.
Competing with G is the discriminator D trained to distinguish samples $\tilde{x} \sim p_{model}(x)$ from real data $x \sim p_{data}(x)$.
Figure 7 summarizes the described architecture with the respective gradient flow used to

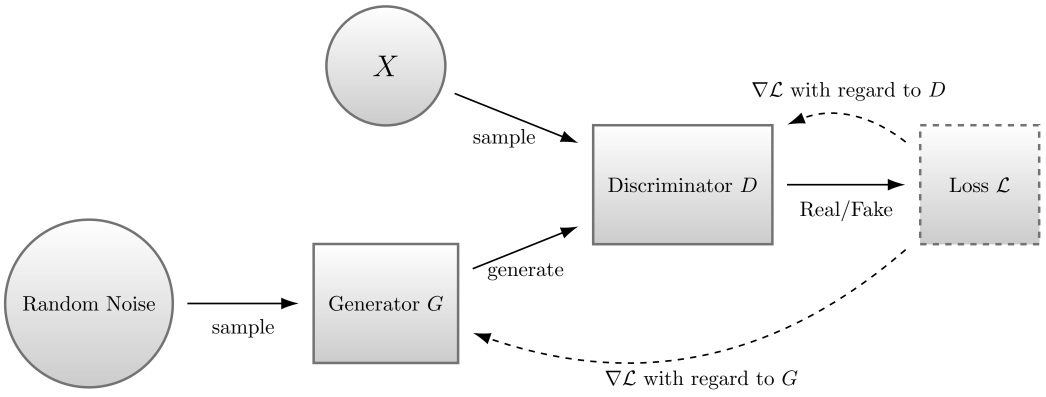


Figure 7: Architecture used for generative adversarial networks

The above figure illustrates the computational graph and gradient flow in the GAN framework. X denotes the dataset containing real-world data sampled from p_{data}, which we aim to model. Solid arrows indicate the forward pass in the model while dashed arrows denote the gradient flow during a backward pass.

update G and D.

4.1.1 Generator Network

The generator G in the GAN framework tries to fool the discriminator D by producing samples which are indistinguishable from training data. This process can formally be written as minimizing $\mathcal{L}_g$ in equation 12.

$$\mathcal{L}_g = \min \log \Big(1 - D(G(z))\Big) \tag{12}$$

However, for gradient-based learning $\mathcal{L}_g$ provides poor gradient during learning where the discriminator is able to reject samples from the generator with high confidence. Therefore is Equation 13 advised to be used instead of Equation 12 [10].

$$\mathcal{L'}_g = \max \log D(G(z)) \tag{13}$$

Note that this loss function is fully differentiable which means it can be trained with backpropagation.

4.1.2 Discriminator Network

The objective of the discriminator network D is to distinguish the generated samples produced by G from the training set samples. This process can be formalized to maximizing the cost function $\mathcal{L}_d$ in equation 14.

$$\mathcal{L}_d = \max \log \big(\log D(x) + \log(1 - D(G(z))\big) \tag{14}$$

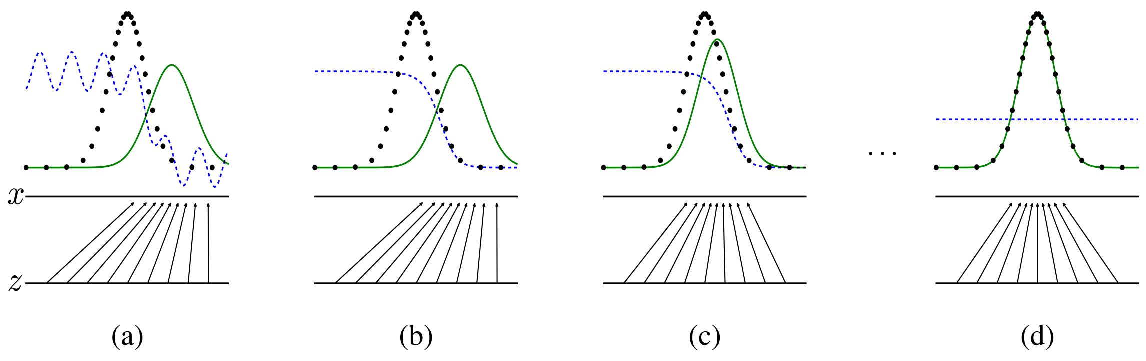


Figure 8: Learning a GAN on a one-dimensional distribution from the original paper[10]

The line named z denotes the noise prior, while the arrows to the line indicated by x is the transformation learned by the generator network. Three different distributions are showed, the solid line represents the generated sample in each step, while the dots are sampled from the data distribution $p_{data}(x)$. The dashed line shows the decision boundary with which probability the discriminator can distinguish between real and fake data.
(a) Prior to training when both D and G are poor and p_{model} does not match p_{data}.
(b) After a step of learning D with fixed G, the discriminator is able to distinguish fake from real samples easily as indicated by the decision boundary.
(c) After a step of learning G with fixed D has the generator learned to produce samples more similar to $p_{data}(x)$.
(d) The model has reached the nash equilibrium where $p_{model} = p_{data}$ and the discriminator can only distinguish between real and fake samples with $p = 0.5$.

4.2 Training

Combining the two objectives into one value function yields equation 15 which is used to define the minmax-game played by the two networks.

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{data}}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z)))] \tag{15}$$

The GAN framework can be fully trained with stochastic backpropagation due to $V(G,D)$ being differentiable with G and D implemented as (deep) neural nets. Figure 8 illustrates the ideal training procedure for learning of an one-dimensional distribution. Training will switch between training the generator and discriminator, where the other party is fixed respectively.

Using a model with infinite training time as well as infinite capacity in both generator and discriminator allows formulating the nash equilibrium in the generative adversarial network. For any given generator G and data points $x \sim p_{data}(x)$, the optimal discriminator D is given by equation 16.

$$D^*_G(x) = \frac{p_{data}(x)}{p_{data}(x) + p_{model}(x)} \tag{16}$$

4.3 Stability and Performance

In the common case where both function approximators G and D are implemented using deep neural networks, GANs have been shown to be difficult to train using gradient descent. In fact, there is no guarantee that GANs will converge in the case of using non-convex functions to represent G or D [9, p. 700-701].

Acknowledging these problems, Salimans et al.[33] have studied the stability and performance of GAN in detail, resulting in a number of practical methods improving performance of the training procedure of GANs. We will take a look at a selection of these methods in the following, focussing on deep neural networks as approximators in both D and G.

4.3.1 Freeze Learning

One of the main issues prominent during GAN training is the disparity in performance between G and D. Most of the time during training will one network outperform the other player by such great margin that the gradients from the stronger player will vanish resulting in even slower training for the weaker network. Thus one of the common practices[33][27] for learning GANs is to freeze one network and prevent it from learning until the other player has caught up.

4.3.2 Label Smoothing

Another rather basic idea to improve gradient flow is to replace the binary classification from the discriminator with smoothed values 0.1/0.9 or $\epsilon/1-\epsilon$[33]. This helps the generator with an informative gradient to improve learning performance. Additionally, smoothed labels have been shown to reduce the attack surface of neural networks to adversarial examples [40].

4.3.3 (Virtual) Batch normalization

Batch normalization (BN) [16] is a recent technique to boost performance in deep neural networks addressing the *internal covariate shift* by normalizing minibatches. Essentially, BN works by normalizing each sample from a minibatch to the statistics of the whole batch. To improve training performance of GANs, a modified version of batch normalization called virtual batch normalization has been proposed [33] for training of the generator. Instead of normalizing each input x to their corresponding minibatch, one reference batch is chosen prior to training which will be used to normalize all further input.

4.3.4 Feature Matching

In computer vision, features are often used to identify relevant pieces of information in images for recognition and classification. With the recent advances in classification tasks performed by deep convolutional networks, automatically learned features have been a large part of that success. Feature matching is an technique designed to improve the learning procedure of GANs where $\mathcal{L}_g$ is not directly maximized but where the discriminator also learns on intermediate layer activation outputs. Using this modified procedure aids the generator to output data matching the statistics of the input data more closely.

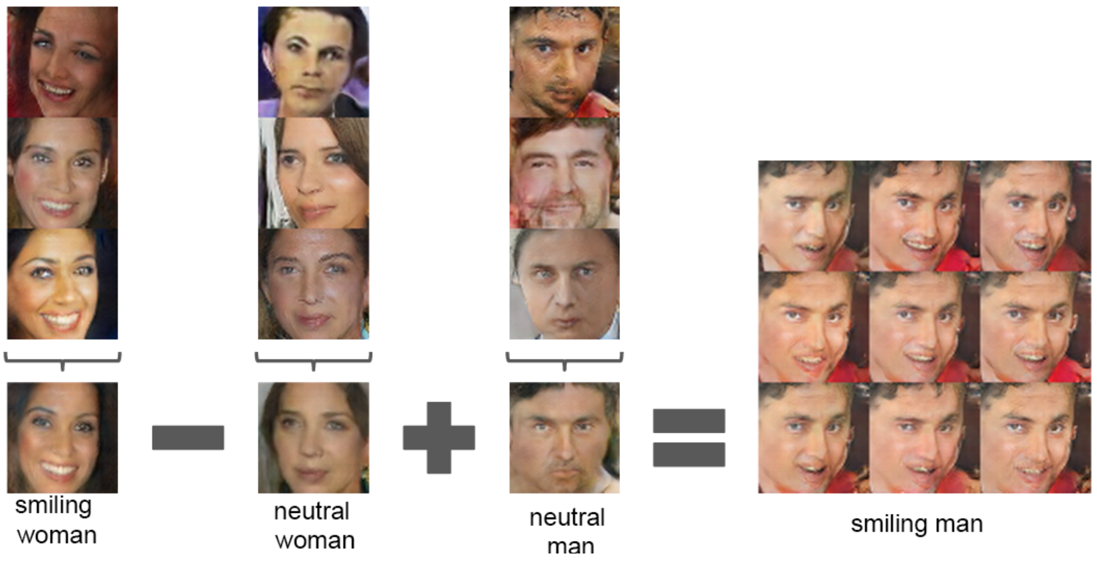


Figure 9: Vector arithmetic performed in latent space z, from DCGAN paper[27]

4.4 Extensions

4.4.1 Deep Convolutional GAN

Deep convolutional networks[21] have yielded impressive results for image classification [19] and have arguably been one of the main reasons for the recent success of deep learning.

The discriminator in the GAN model implements a binary classification task and the performance of the generator is heavily dependent on the discriminator, as the training procedure uses the discriminator in the objective function $\mathcal{L}_g$ in equation 12 and 13. This leads to the suggestion that using deep convolutional networks helps boost the model performance. In fact is the deep convolution GAN proposed by Radford et al.[27] one of the first GAN variants which result in visually pleasing results.

In order to use convolutional networks in the generator and discriminator, fractionally strided convolutions (also called deconvolution [42]) have been used to basically upsample the image. The exact details of the architecture is beyond the scope of this article, however in addition to the impressive generative performance the authors have shown that the model has learned to produce representational structures in latent space. Furthermore, as shown in Figure 9 they were able to perform vector arithmetic in the latent space with results indicating that their model was able to arrange the latent space in a useful way.

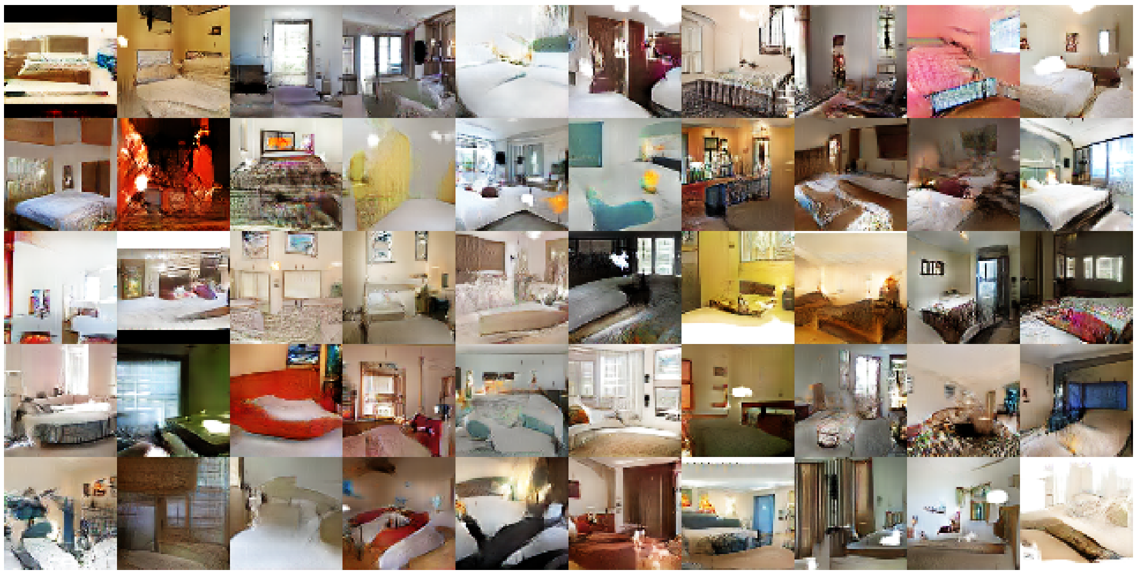


Figure 10: Bedroom scenes generated by the DCGAN model, from the original paper[6]

4.4.2 Laplacian Generative Adversarial Networks

Laplacian Generative Adversarial Networks (LAPGAN) [6] are an extension to GANs applied to images using laplacian pyramids [5]. Instead of using a single generator network, LAPGAN uses a series of deep convolutional networks as generators where each generator receives the output of the previous generator.

Training is heavily modified from the regular GAN framework. In each iteration a training sample is drawn from the dataset and a downsamples version from it is computed. The discriminator receives now either the computed high-pass image (from the original image and the down-then-up sampled image) or an high-pass image from the generator. These steps are repeated for other steps, downscaling the image. In the last stage, a traditional generator and discriminator is used for the 8×8 image.

Sampling uses a conditional GAN to produce high-resolution images through different stages of deep generator networks.

Results As GANs have no direct measurement of the quality of generations due to their lack of unified objective function, human evaluation of the authenticity of generated images is often used for qualitively evaluation. LAPGAN has pushed the state of the art significantly, with generated images which are high-resolution in comparison to earlier methods and arguably high variety of generated images as shown in Figure 10. Furthermore, humans have mistaken the generated images for real images 40% of the time.

5 Summary

In this article, we have taken a look at various directed models which perform generative modeling using deep neural networks.
After a short overview over traditional generative models and models not covered in this article, we have taken a look at a wide area of applications using generative models.

As one of the first basic generative models, the sigmoid belief net lays the foundation for directed generative models and displays its weaknesses during inference and training. The variational autoencoders have been discussed in more detail using variational inference to turn the inference problem into an optimization problem and solving it there. VAE and its various extensions have been shown to be a flexible tool applicable for different tasks.
Generative adversarial networks introduced in Section 4 uses a game-theoretic approach to generative modeling. In recent years, GANs have been extended of which we discussed a subset.

Deep generative models have shown the capability to learn rich internal representation of high-dimensional data in order to generate samples using recent advances in deep learning. Going further, as intelligent agents will need to have accurate and generalized internal representations inferred through sensory input to reason about the world, generative models are likely to be the cornerstones of more advanced intelligent systems in the future.

Especially interesting is the case of unsupervised learning, due to vast amount of freely available data on the web and in images, video and audio. Deep learning has been mostly focussed on supervised algorithms, however these methods require most labelled datasets which are often not feasible to obtain.

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

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

[3] Christopher M. Bishop. *Pattern Recognition and Machine Learning.* Springer, 2006.

[4] Yuri Burda, Roger Grosse, and Ruslan Salakhutdinov. Importance weighted autoencoders. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2016.

[5] Peter Burt and Edward Adelson. The laplacian pyramid as a compact image code. *IEEE Transactions on communications*, 31(4):532–540, 1983.

[6] Emily L Denton, Soumith Chintala, Arthur Szlam, and Rob Fergus. Deep generative image models using a laplacian pyramid of adversarial networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 1486–1494. Curran Associates, Inc., 2015.

[7] L. A. Gatys, A. S. Ecker, and M. Bethge. Texture synthesis using convolutional neural networks. In *Advances in Neural Information Processing Systems 28*, May 2015.

[8] Zoubin Ghahramani and Sam Roweis. Probabilistic models for unsupervised learning. *NIPS Tutorial*, 1999.

[9] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep learning. Book in preparation for MIT Press, 2016.

[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] Karol Gregor, Ivo Danihelka, Alex Graves, Danilo Rezende, and Daan Wierstra. Draw: A recurrent neural network for image generation. In *Proceedings of The 32nd International Conference on Machine Learning*, pages 1462–1471, 2015.

[12] Richard HR Hahnloser, Rahul Sarpeshkar, Misha A Mahowald, Rodney J Douglas, and H Sebastian Seung. Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. *Nature*, 405(6789):947–951, 2000.

[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] Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. In *Advances in Neural Information Processing Systems*, pages 4565–4573, 2016.

[15] Rein Houthooft, Xi Chen, Yan Duan, John Schulman, Filip De Turck, and Pieter Abbeel. Vime: Variational information maximizing exploration. In *Advances in Neural Information Processing Systems*, pages 1109–1117, 2016.

[16] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of The 32nd International Conference on Machine Learning*, pages 448–456, 2015.

[17] Diederik P Kingma, Shakir Mohamed, Danilo Jimenez Rezende, and Max Welling. Semi-supervised learning with deep generative models. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 3581–3589. Curran Associates, Inc., 2014.

[18] D.P Kingma and M. Welling. Stochastic gradient vb and the variational auto-encoder. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2014.

[19] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[20] S. Kullback and R. A. Leibler. On information and sufficiency. *Ann. Math. Statist.*, 22(1):79–86, 03 1951.

[21] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, Winter 1989.

[22] Soroush Mehri, Kundan Kumar, Ishaan Gulrajani, Rithesh Kumar, Shubham Jain, Jose Sotelo, Aaron Courville, and Yoshua Bengio. Samplernn: An unconditional end-to-end neural audio generation model. *Accepted at ICLR 2017*, 2016.

[23] Radford M. Neal. Connectionist learning of belief networks. *Artificial Intelligence*, 56(1):71–113, jul 1992.

[24] Martin J. Osborne. *A Course in Game Theory*, chapter 2, page 14. 1994.

[25] J. Pearl. Bayesian networks: A model of self-activated memory for evidential reasoning. Technical Report CSD-850021, R-43, UCLA Computer Science Department, June 1985.

[26] Lawrence R Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[27] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2016.

[28] Scott Reed, Zeynep Akata, Xinchen Yan, Lajanugen Logeswaran, Bernt Schiele, and Honglak Lee. Generative adversarial text to image synthesis. In *International Conference on Machine Learning (ICML)*, 2016.

[29] Danilo J. Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In Tony Jebara and Eric P. Xing, editors, *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 1278–1286. JMLR Workshop and Conference Proceedings, 2014.

[30] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1, 1988.

[31] Stuart J. Russell and Peter Norvig. *Artificial Intelligence - A Modern Approach.* Pearson, 2009.

[32] Ruslan Salakhutdinov and Geoffrey E Hinton. Deep boltzmann machines. In *AISTATS*, volume 1, page 3, 2009.

[33] Tim Salimans, Ian J. Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. *CoRR*, abs/1606.03498, 2016.

[34] Mike Schuster. Better generative models for sequential data problems: Bidirectional recurrent mixture density networks. In *NIPS*, pages 589–595, 1999.

[35] Paul Smolensky. Information processing in dynamical systems: Foundations of harmony theory. Technical report, COLORADO UNIV AT BOULDER DEPT OF COMPUTER SCIENCE, 1986.

[36] Kihyuk Sohn, Honglak Lee, and Xinchen Yan. Learning structured output representation using deep conditional generative models. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 3483–3491. Curran Associates, Inc., 2015.

[37] Casper Kaae Sønderby, Jose Caballero, Lucas Theis, Wenzhe Shi, and Ferenc Huszár. Amortised map inference for image super-resolution. *Accepted at ICLR 2017*, 2017.

[38] L. Theis, S. Gerwinn, F. Sinz, and M. Bethge. In all likelihood, deep belief is not enough. *Journal of Machine Learning Research*, 12:3071–3096, Nov 2011.

[39] Carl Vondrick, Hamed Pirsiavash, and Antonio Torralba. Generating videos with scene dynamics. In D. D. Lee, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances In Neural Information Processing Systems 29*, pages 613–621. Curran Associates, Inc., 2016.

[40] David Warde-Farley and Ian Goodfellow. Adversarial perturbations of deep neural networks. In Tamir Hazan, George Papandreou, and Daniel Tarlow, editors, *Perturbations, Optimization, and Statistics*, chapter 11. MIT Press, 2016.

[41] Jiajun Wu, Chengkai Zhang, Tianfan Xue, William T Freeman, and Joshua B Tenenbaum. Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling. In *Neural Information Processing Systems (NIPS)*, 2016.

[42] Matthew D Zeiler, Dilip Krishnan, Graham W Taylor, and Rob Fergus. Deconvolutional networks. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 2528–2535. IEEE, 2010.