Université de Montréal

Generative Models for Natural Images

par:
Faruk Ahmed

Directeur de recherche:
Aaron Courville

Mémoire présenté à la Faculté des arts et des sciences
en vue de l'obtention du grade de Maître ès sciences (M.Sc.)
en informatique

Institut des algorithmes d'apprentisage de Montréal
Département d'informatique et de recherche opérationnelle

Août, 2017

Résumé

Nous traitons de modèles génératifs construits avec des réseaux de neurones dans le contexte de la modélisation d'images. De nos jours, trois types de modèles sont particulièrement prédominants: les modèles à variables latentes, tel que l'auto-encodeur variationnel (VAE), les modèles autorégressifs, tel que le réseau de neurones récurrent pixel (PixelRNN), et les modèles génératifs antagonistes (GANs), qui sont des modèles à transformation de bruit entrainés à l'aide d'un adversaire. Cette thèse traite de chacun de ces modèles.

Le premier chapitre couvre la base des modèles génératifs, ainsi que les réseaux de neurones profonds, qui constituent la technologie principalement utilisée à l'heure actuelle pour l'implémentation de modèles statistiques puissants.

Dans le deuxième chapitre, nous implémentons un auto-encodeur variationnel avec un décodeur auto-régressif. Cela permet de se libérer de l'hypothèse d'indépendance des dimensions de sortie du décodeur variationnel, en modélisant une distribution jointe traçable à la place, et de doter le modèle auto-régressif d'un code latent. De plus, notre implémentation a un coût computationnel significativement réduit, si on le compare à un modèle purement auto-régressif ayant les mêmes hypothèses de modélisation et la même performance. Nous décrivons l'espace latent de façon hiérarchique, et montrons de manière qualitative la décomposition sémantique des causes latente induites par ce design. Finalement, nous présentons des résultats obtenus avec des jeux de données standards et démontrant que la performance de notre implémentation est fortement compétitive.

Dans le troisième chapitre, nous présentons une procédure d'entrainement améliorée pour une variante récente de modèles génératifs antagoniste. Le «Wasserstein GAN» minimise la distance, mesurée avec la métrique de Wasserstein, entre la distribution réelle et celle générée par le modèle, ce qui le rend plus facile à entrainer qu'un GAN avec un objectif minimax. Cependant, en fonction des paramètres, il présente toujours des cas d'échecs avec certain modes d'entrainement. Nous avons découvert que le coupable est le coupage des poids, et nous le remplaçons par une pénalité sur la norme des gradients. Ceci améliore et stabilise l'entrainement, et ce sur différents types du paramètres (incluant des modèles de langue sur des données discrètes), et permet de générer des échantillons de haute qualités sur CIFAR-10 et LSUN bedrooms.

Finalement, dans le quatrième chapitre, nous considérons l'usage de modèles génératifs modernes comme modèles de normalité dans un cadre de détection hors-distribution «zero-shot». Nous avons évalué certains des modèles précédemment présentés dans la thèse, et avons trouvé que les VAEs sont les plus prometteurs, bien que leurs performances laissent encore un large place à l'amélioration. Cette partie de la thèse constitue un travail en cours.

Nous concluons en répétant l'importance des modèles génératifs dans le développement de l'intelligence artificielle et mentionnons quelques défis futurs.

Mots clés Apprentissage automatique, réseaux de neurones, apprentissage de représentations profondes, modèles génératifs

Abstract

We discuss modern generative modelling of natural images based on neural networks. Three varieties of such models are particularly predominant at the time of writing: latent variable models such as *variational autoencoders* (VAE), autoregressive models such as *pixel recurrent neural networks* (PixelRNN), and *generative adversarial networks* (GAN), which are noise-transformation models trained with an adversary. This thesis touches on all three kinds.

The first chapter covers background on generative models, along with relevant discussions about deep neural networks, which are currently the dominant technology for implementing powerful statistical models.

In the second chapter, we implement variational autoencoders with autoregressive decoders. This removes the strong assumption of output dimensions being conditionally independent in variational autoencoders, instead tractably modelling a joint distribution, while also endowing autoregressive models with a latent code. Additionally, this model has significantly reduced computational cost compared to that of a purely autoregressive model with similar modelling assumptions and performance. We express the latent space as a hierarchy, and qualitatively demonstrate the semantic decomposition of latent causes induced by this design. Finally, we present results on standard datasets that demonstrate strongly competitive performance.

In the third chapter, we present an improved training procedure for a recent variant on generative adversarial networks. Wasserstein GANs minimize the Earth-Mover's distance between the real and generated distributions and have been shown to be much easier to train than with the standard minimax objective of GANs. However, they still exhibit some failure modes in training for some settings. We identify weight clipping as a culprit and replace it with a penalty on the gradient norm. This improves training further, and we demonstrate stability on a wide variety of settings (including language models over discrete data), and samples of high quality on the CIFAR-10 and LSUN bedrooms datasets.

Finally, in the fourth chapter, we present work in development, where we consider the use of modern generative models as normality models in a zero-shot out-of-distribution detection setting. We evaluate some of the models we have discussed previously in the thesis, and find that VAEs are the most promising, although their overall performance leaves a lot of room for improvement.

We conclude by reiterating the significance of generative modelling in the development of artificial intelligence, and mention some of the challenges ahead.

Keywords Machine learning, neural networks, deep learning, generative models.

Acknowledgements

The past two years of graduate studies have been an exciting journey that wouldn't have been as positively fulfilling without the wonderful people in my life. I am very very grateful to you all.

I thank my advisor, Aaron Courville, for his supervision and mentorship, always sharing his wisdom and good humour.

I am especially indebted and grateful to my friend and collaborator Ishaan Gulrajani, without whom this particular thesis would not have existed. I would also like to thank all the clever people I have had the good fortune to work with, in particular Martín Arjovsky and Kundan Kumar, for all the patient and enlightening discussions they've indulged me with.

A special thank you to Stanisław Antol and Pauline Tranchant for excellent feedback on the draft. Thanks to Pauline Tranchant and César Laurent for translating the abstract into *français.*

Thanks to César Laurent for carrying me on his shoulders and doing my groceries when I sprained my ankle. Thanks to Samira Shabanian, Mohammad Pezheshki, and Mohammad Havaei for being such helpful friends in times of need.

Thanks to Céline Begin for help with administrative processes.

A final *Thank You!* to all the folk I share my life with, for providing me with such marvellous company.

Contents

List of Figures

List of Tables

List of Abbreviations

wrt	**w**ith **r**espect **t**o
iid	**i**ndependent and **i**dentically **d**istributed
MLP	**m**ulti-**l**ayer **p**erceptron
CNN	**c**onvolutional **n**eural **n**etwork
VAE	**v**ariational **a**uto**e**ncoder
RNN	**r**ecurrent **n**eural **n**etwork
LSTM	**l**ong **s**hort-**t**erm **m**emory
GAN	**g**enerative **a**dversarial **n**etwork
DCGAN	**D**e**c**onvolutional **GAN**
LSGAN	**L**east-**S**quares **GAN**
WGAN	**W**asserstein **GAN**
WGAN-GP	**WGAN** with **g**radient **p**enalty
IAF	**i**nverse **a**utoregressive **f**low
ALI	**a**dversarially **l**earned **i**nference
BN	**b**atch **n**ormalization
ReLU	**re**ctified **l**inear **u**nit
SGD	**s**tochastic **g**radient **d**escent
BPTT	**b**ack**p**ropagation **t**hrough **t**ime
MSE	**m**ean **s**quared **e**rror
AUPR	**a**rea **u**nder the **p**recision-**r**ecall curve
NLL	**n**egative **l**og-**l**ikelihood
KL	**K**ullback-**L**eibler
JSD	**J**ensen-**S**hannon **d**ivergence
ELBO	**e**vidence **l**ower **bo**und
AI	**a**rtificial **i**ntelligence
GPU	**g**raphics **p**rocessing **u**nit

List of Symbols

Symbol	*Meaning*	*Notes*
$*$	Convolution operator	$\boldsymbol{K} * \boldsymbol{x}$ denotes $\boldsymbol{x}$ convolved with the kernel $\boldsymbol{K}$
$\odot$	Hadamard operator	$\boldsymbol{x}_1 \odot \boldsymbol{x}_2$ denotes element-wise multiplication of $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$
$\boldsymbol{A}^T$	Matrix transpose	Transpose of the matrix $\boldsymbol{A}$
$\boldsymbol{x}$	Observed random variable	In this thesis, most often an image
$\boldsymbol{h}$	Latent variable	Unobserved causes in latent variable models
$\boldsymbol{z}$	Noise variable	Sampling noise in noise-transformation models
$\boldsymbol{\theta}$	Parameters of a model	Typically a vector; individual elements are θ
$\boldsymbol{\phi}$	Variational parameters	Used to denote parameters of approximate posteriors
$\boldsymbol{I}$	Identity matrix	The dimensions are inferred from context

Chapter 1

Introduction

1.1 Generative models

A *generative model* can be considered an imaginative machine. Given access to a set of samples from an unknown target distribution, a generative model provides a mechanism for producing samples from the distribution learned by the model. We would like the samples to be:

- **similar**, in the sense of sharing structure, and not trivial memorizations of observed data,
- **plausible**, in the sense of not breaking the rules of the world as implicit in the observations; for example, a bird with metal wings is implausible (assuming that the model has never observed such an occurence), and
- **representative**, in the sense of accurately reflecting the distribution implicit in the observed data; for example, capturing a subset of the distribution and dropping the rest would not be representative.

There is an enormous amount of implicit world-knowledge that surrounds us, which we have imbibed without directly interacting with everything in a way that gives us precise feedback about their nature. Learning how to group observations by similarities of patterns and structures, without access to any particular semantic categorization of the observations or interactive feedback, is referred to as *unsupervised learning.* We would like computers to be able to learn about their environment in an equally powerful fashion, if not more, and a very key step towards this goal would be the ability to construct meaningful representations of the environment by simply observing the world. Evolution has endowed us with advantages, such as the structure of the eye and neurons, and learning principles followed in the brain, that make it easier for us to learn in a largely unsupervised manner. Similarly, we provide our computational models with specialized forms, and instill in them procedures that govern the nature of learning in their mechanism.

1.2 Why is generative modelling useful?

Models that generate meaningful samples from a learned distribution about observed data, without having trivially memorized the observations or generating minor perturbations, hold in them an implicit understanding of their observations. This understanding can be tapped into for exploring semantic dimensions in the space of learned representations, and for inferring deeper structure and hidden causal factors of the data. Higher level representations would encode compact "concepts" which can be a ground for reasoning and making decisions. Another potential use for such a model could be detecting events that are unexpected or out-of-distribution. We might also use generative models for learning conditional distributions where we add some semantic/task-specific information as an extra input (an example of such a task would be generating an image conditioned upon a text description of it). Being able to generate plausible data also means we can simulate possible scenarios in a stochastic environment, conditioned on an initial state and actions, which would enable us to perform exhaustive planning in complicated situations and environments, before taking costly or potentially adverse actions.

Some specific examples of applications are the following:

- **data imputation:** A model that can learn a probability distribution over observations can be used to fill in missing portions of input data, by sampling from that distribution after conditioning upon the visible input. This has been used, for example, to in-paint an image, or denoise it, or enhance the resolution [92, 43, 33].
- **semi-supervised classification:** Labelled data is typically more expensive to acquire. However, unlabelled data is usually cheaply available and easy to mine, which can be used to construct better decision boundaries than would have been possible using the labelled data alone. One approach for this is by constructing generative models, considering the label as a latent variable in the model, as shown in [39].
- **data compression:** Data compression typically involves an encoder/decoder setup. Generative models can be used in the decoder to achieve far higher compression rates than traditional compression schemes (in a setup where conceptual rather than exact reconstruction is the end goal), as demonstrated in Santurkar, Budden, and Shavit [75]. Better compression makes communication more efficient.
- **translation across domains:** Conditional generative models have been used for tasks involving translation across domains. For example, Oord et al. [58] demonstrate application of generative modelling for performing text-to-speech synthesis, and Reed et al. [65] perform text-to-image synthesis.
- **representation learning:** Generative models with latent variables designed to learn encodings that are disentangled across dimensions have been shown to capture useful representation spaces that lend themselves well to downstream tasks such as *zero-shot inference* (which entails inferring attributes of previously unseen objects), as in [28].

In addition to these, generative models have been used in other diverse areas such as astronomy [66] and drug discovery [25]. The ability to generate content has obvious uses in creative fields, such as generating stylistic images, dangerous stunts in movies, and other digital artistic creations [21].

1.3 Categorization of generative models

In this section we shall discuss two categorizations of generative models: one, based on their structure, and another, from a probabilistic viewpoint.

1.3.1 Fully observed, latent variable, and noise-transformation models

A broad classification of modern generative models based on the random variables constituting the model would place them into three categories: fully observed models, latent variable models, and noise-transformation models [52].

Fully observed models Data is directly modelled by interactions among random variables, such that every random variable is observed, and no unobserved random variable is introduced into the model. These models are typically specified by a number of conditional distributions (such as conditioning sets of pixels upon other sets in a natural image). We can formulate such models by a directed graph, by conditioning some random variables on others, in which case learning simply consists of counting statistics. However, we would need to assign an ordering of the variables for which there might not always be an obvious choice. For example, in a natural image, usually we cannot tell which pixel is a "primary cause" of the entire image. An undirected graph formulation (commonly called a *Markov Random Field*) removes the ordering requirement, but learning in such models requires computing intractable normalizing constants, and sampling is slow (either sequential, or requires iterative steps of applying a transition operator).

Latent variable models Given a data distribution, there might be very complex interdependencies between the dimensions. Modelling all of these interdependencies is costly, and makes training much harder in a high dimensional data space, since every dimension has to keep every other dimension happy. A simple way to make this easier is to assume there is a (low-dimensional) *latent variable* $\boldsymbol{z}$ that induces global structure on the samples, thus freeing us from modelling complex interactions among the observed random variables in order to capture high level structure, since the observed random variables are conditionally independent given the latent variables. For example, to generate a natural image, the latent variable might encode hidden "causes" behind the image such as the object categories, their positions and orientations, etc., and then a generator only needs to decode this high level information into a set of pixels that visually represents it. The variable $\boldsymbol{z}$ is referred to as *latent* because we never see these variables in the training data, only the observed random variables $\boldsymbol{x}$ are seen. To deduce our model's latent $\boldsymbol{z}$ corresponding to an observed $\boldsymbol{x}$ we need to perform *inference*. If we assume that the model is parameterized by $\boldsymbol{\theta}$, we would like to train the model such that on average, the samples produced by the model by sampling from a prior over $\boldsymbol{z}$, $p(\boldsymbol{z})$, would match the data distribution. Formally, we would like to maximize

$$p(\boldsymbol{x}) = \int p(\boldsymbol{x}|\boldsymbol{z};\boldsymbol{\theta})p(\boldsymbol{z})d\boldsymbol{z}, \tag{1.1}$$

which is referred to as the maximum likelihood training principle for latent variable models.

An advantage provided by latent variable models is that we can use the inferred latent variables corresponding to a given observation for downstream tasks such as representation learning or compression (see section 1.2).

Noise-transformation models In this case, we learn to transform noise (from some simple noise distribution such as uniform or Gaussian) into a sample in data space through a function g parameterized by $\boldsymbol{\theta}$:

$$\boldsymbol{z} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I}), \boldsymbol{x} = g_{\boldsymbol{\theta}}(\boldsymbol{z}). \tag{1.2}$$

The key difference with latent variable models is that, by design, the noise distribution is not intended to model latent factors [1] (although the learned parametric mapping might induce some notion of causal factors of variation, which may be revealed through visualizations such as interpolations between samples). The distribution being modelled can be expressed, at least in theory, by following the change of variables rule (assuming invertibility):

$$p(\boldsymbol{x}) = p(\boldsymbol{z})\left|\det\frac{\partial g}{\partial \boldsymbol{z}}\right|^{-1}, \tag{1.3}$$

and this distribution may not be accessible if g is not invertible. Sampling from such models is very straightforward, but optimization and evaluation is typically challenging mostly due to not being able to maintain invertibility.

[1] Another point of difference with latent variable models is that computing the data likelihood typically does not involve marginalizing over a set of latent variables.

In the next section, we shall look at a categorization of generative models from a probabilistic viewpoint, in terms of whether they have an explicit specification of a modelled density or not.

1.3.2 Explicit vs. implicit models

Diggle and Gratton [18] discuss the categorization of statistical models into being *implicit* or *prescribed* (which we shall refer to as *explicit*).

An *explicit statistical model* of an observed set of random variables $\boldsymbol{x}$ implies a parametric specification of the distribution of $\boldsymbol{x}$ with parameters $\boldsymbol{\theta}$, which leads to the definition of a log-likelihood function $\mathcal{L}$ expressed as

$$\mathcal{L}(\boldsymbol{x}|\boldsymbol{\theta}) = \log p_{\boldsymbol{\theta}}(\boldsymbol{x}), \tag{1.4}$$

where $p_{\boldsymbol{\theta}}$ is a class of distributions parameterized by $\boldsymbol{\theta}$. Inference problems of interest are estimation of the parameters $\boldsymbol{\theta}$, and goodness-of-fit tests for the distribution function p. Explicit statistical models are a natural choice for density modelling.

An *implicit statistical model* is one where a stochastic data generating process defines the model. This is typically a transformation model, meaning that the process involves sampling a random variable from a distribution easy to sample from, $\boldsymbol{z} \sim p(\boldsymbol{z})$, which is then transformed into a sample in the domain of interest through a deterministic mapping $g_{\boldsymbol{\theta}}(\boldsymbol{z}) : \boldsymbol{z} \in \mathcal{R}^d \rightarrow \boldsymbol{x} \in \mathcal{R}^n$, where $\boldsymbol{\theta}$ are the parameters of the function g. Implicit statistical models are a more natural choice when we want to simulate data from a mechanistic understanding of a system that gives rise to observations.

Log-likelihood in implicit models As noted by Mohamed and Lakshminarayanan [53], in general, implicit models have a unique log-likelihood function associated with them, $p_{\boldsymbol{\theta}}(\boldsymbol{z})$, which can be written as the derivative of the cumulative distribution function via the transformed density to $p(\boldsymbol{z})$,

$$p_{\boldsymbol{\theta}}(\boldsymbol{x}) = \frac{\partial}{\partial \boldsymbol{x}} \int_{g_{\boldsymbol{\theta}}(\boldsymbol{z}) \leq \boldsymbol{x}} p(\boldsymbol{z}) d\boldsymbol{z}, \tag{1.5}$$

which we can recover, at least in theory, if the function g is invertible (so we can find the regions where $g_{\boldsymbol{\theta}}(\boldsymbol{z}) \leq \boldsymbol{x}$), or if $d = n$ with finite and easily found roots of the determinant of the Jacobian (so that the inverse function theorem can be used). A log-likelihood would not exist in cases where a valid probability density is not admitted. This can happen, for example, when $n > d$ so that all of $g_{\boldsymbol{\theta}}(\boldsymbol{z})$ lies on a lower $d-$dimensional manifold in $\mathcal{R}^n$. As discussed by Arjovsky and Bottou [2], in this case, the random variable $\boldsymbol{x}$ has non-zero probability on a lower dimensional manifold in $\mathcal{R}^n$, which is a measure zero set, and the Radon-Nikodym theorem implies that this precludes a valid probability density (*wrt* the Lebesgue measure on the whole space), hence also a log-likelihood function.

1.4 Evaluation

To test how well an explicit model fits the data, the simplest evaluation measure is the log-likelihood on a held out test set. As discussed above, an implicit model usually does not make a tractable log-likelihood accessible to us, and so we typically have to devise ways to evaluate the model. Since the model involves a mechanism for generating samples easily, we can use kernel density estimation [62] for approximting a generating distribution, and look up likelihood of the test set under this distribution. However, this method loses accuracy in higher dimensions, as discussed in Theis, Oord, and Bethge [82].

Moreover, as Theis, Oord, and Bethge [82] also point out, log-likelihood measures do not appear to correspond well to visual sample quality for modern generative models, which is the most common way for evaluating models for natural images. Surrogates for this assessment have been developed, the most popular being the Inception Score [73], which rewards a model for producing samples that a classifier predicts confident class labels for, and penalizes the model if there is less variance in the predicted classes across a set of samples.

It is not clear if there exists one best way to evaluate or even one best optimization objective for generative models. The best answer might be that it is goal-oriented, and we would need to define specific criteria and metrics tailored to the intended application in a downstream task. For example, if the purpose is to generate samples from a (conditional) distribution to perform image super-resolution (where the goal is to enhance the resolution of an input image), we would like our samples to look as visually appealing and close to real world images as possible, whether or not the model gives us calibrated likelihoods. If we want a model that we shall use to measure the likelihood of test data under a learned distribution, we are only concerned with meaningful likelihood numbers, and are willing to sacrifice on visual quality of samples. As another example, if we want to create a latent variable model for the purpose of using the inferred latent variables for some other task, optimizing for log-likelihood does not guarantee usefulness of the latent variables, since $p(\boldsymbol{x}, \boldsymbol{z}) = p(\boldsymbol{x})p(\boldsymbol{z}|\boldsymbol{x})$, and optimizing for the marginal likelihood $p(\boldsymbol{x})$ says nothing about usefulness of the posterior $p(\boldsymbol{z}|\boldsymbol{x})$, and a powerful model might well choose to ignore the latent space completely, with a strong reconstruction model $p(\boldsymbol{x}|\boldsymbol{z})$ and matching exactly a prior on $\boldsymbol{z}$.

In the following sections, we shall provide background information about the models that we are concerned with. In particular, we will review the three classes of generative models for natural images that are currently most prevalent: the variational autoencoder, pixel-recurrent/convolutional neural networks, and generative adversarial networks. Of these, variational autoencoders and pixel recurrent neural networks are explicit statistical models, while generative adversarial networks are implicit models. Variational autoencoders are latent variable models that provide us with a lower bound to the log-likelihood, while pixel recurrent neural networks are fully observed models with a tractable exact log-likelihood, and generative adversarial networks are noise-transformation models which do not make a log-likelihood accessible to us.

1.5 Variational autoencoder

The *variational autoencoder*, introduced concurrently by Kingma and Welling [37] and Rezende, Mohamed, and Wierstra [68], is a powerful latent variable model that is trained by optimizing a lower bound to the intractable marginal data likelihood $p(\boldsymbol{x})$.

We shall first briefly review *variational inference*, which is the fundamental principle behind the model, and then discuss how this is done with the help of *deep neural networks*, which are currently the most powerful computational tools for modelling conditional distributions.

1.5.1 Training latent variable models with variational inference

Recall from Equation 1.1 that given a latent variable model with latent variables $\boldsymbol{z}$ and observed variables $\boldsymbol{x}$, we are interested in maximizing the marginal likelihood of the data under our model $p_{\boldsymbol{\theta}}(\boldsymbol{x}) = \int p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})p(\boldsymbol{z})d\boldsymbol{z}$, where our model is parameterized by $\boldsymbol{\theta}$. Simple forms of p, which might allow us to calculate this integral in closed form, are not powerful enough for modelling complex distributions. Hence, it's usually the case that the integral is intractable, and sampling-based approximations typically don't provide good estimates because an extremely high number of samples would be required in order to capture points in the high dimensional latent space that explain the observed data well. A solution would be to use an importance sampling scheme that makes use of an inference mechanism that is more informative about the latent $\boldsymbol{z}$ given an observed $\boldsymbol{x}$. Since we cannot model an exact inference mechanism (due to the latent variables never being observed), we shall perform approximate inference to model the conditional distribution of the latent variable given the observed variable $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$, where q is parameterized by $\boldsymbol{\phi}$. The distribution $q_{\boldsymbol{\phi}}$ is known as the *variational posterior*, and $\boldsymbol{\phi}$ are called the *variational parameters*. The terminology carries over from variational calculus which involves optimizing over a family of functions. [2]

We can think of $q_{\boldsymbol{\phi}}$ as a proposal distribution for computing the integral by importance sampling as follows:

$$\begin{aligned} p_{\boldsymbol{\theta}}(\boldsymbol{x}) &= \int p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})p_{\boldsymbol{\theta}}(\boldsymbol{z})d\boldsymbol{z} && (1.6)\\ &= \int p_{\theta}(\boldsymbol{x}|\boldsymbol{z})\frac{p_{\theta}(\boldsymbol{z})}{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})d\boldsymbol{z}. && (1.7)\end{aligned}$$

Taking logarithms on both sides and applying Jensen's inequality to the integral: [3]

$$\begin{aligned} \log p(\boldsymbol{x}) &= \log \int q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})\frac{p_{\boldsymbol{\theta}}(\boldsymbol{z})}{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}d\boldsymbol{z} && (1.8)\\ &\geq \int q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})\log\left(p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})\frac{p_{\boldsymbol{\theta}}(\boldsymbol{z})}{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\right)d\boldsymbol{z} && (1.9)\\ &= \int q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})d\boldsymbol{z} - \int q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})\log\frac{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}{p_{\theta}(\boldsymbol{z})}d\boldsymbol{z} && (1.10)\\ &= \underbrace{\underbrace{\mathop{\mathbb{E}}_{\boldsymbol{z}\sim q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\Big[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})\Big]}_{\text{reconstruction term}} - \underbrace{KL\big[q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})\ ||\ p_{\boldsymbol{\theta}}(\boldsymbol{z})\big]}_{\text{penalty term}}.}_{\text{variational lower bound}} && (1.11)\end{aligned}$$

where $KL(\cdot,\cdot)$ is the *Kullback-Leibler* (KL) divergence, and is defined between two distributions p and q as $KL(p,q) = \mathbb{E}_p\big[\log p - \log q\big]$.

This lower bound can be optimized *wrt* the parameters $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$, and consists of two components: the first is the *reconstruction error* that measures how well samples from the current approximate posterior $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ explain the observed $\boldsymbol{x}$, and the second term is a *penalty* term that encourages the approximate posterior to match the prior on the latent variables, and can be thought of as a regularizer for the approximate posterior.

Alternatively, we could have arrived at the same variational bound (also called the *evidence lower bound*, or ELBO) by following a different route of derivation, where we start out by trying to minimize the KL divergence between the approximate posterior and the true posterior, which is more aligned with the spirit of variational inference.

[2]This terminology doesn't strictly apply here for us, since we typically do not optimize over functional forms of q, but only over the parameters after "fixing" the functional form.

[3]Recall that Jensen's inequality states that for a concave function g, $g(\mathbb{E}_p[f(x)]) \geq \mathbb{E}_p[g(f(x))]$ and that log is a concave function.

Variational inference typically calls for a *mean-field inference* mechanism, which means that every datapoint is given its own set of variational parameters. This justifies the choice of a simpler family of functions for the approximate posterior, but the reader should keep in mind that we shall eventually use expressive computational models for our approximate posteriors. A simplified version is *amortized inference*, which shares parameters across datapoints. This is useful because every subsequent update to the model parameters can use an improved approximation to the posterior, and learning can be faster. For learning $\boldsymbol{\phi}$ and $\boldsymbol{\theta}$, one can either optimize over each of these alternately, or perform joint optimization. *Stochastic optimization* involves learning parameters over minibatches of data as opposed to a pass over the entire set of data per update.

Variational autoencoders learn through stochastic optimization by performing amortized variational inference using a *deep neural inference network* coupled with a *deep neural generator network*. In the following section we provide some background about deep neural networks and how they are trained before resuming our discussion about how these tools are used to implement variational autoencoders.

1.5.2 Deep neural networks

Deep neural networks are the most powerful technology currently available for performing tasks that require a complex mapping of a high dimensional input signal, such as images/audio/video/text to an output signal, such as classification of objects in an image, actions performed in a video, semantic information from a piece of text, or speech recognition from audio clips. They are also currently the best tool for generative models, and produce compelling samples of images and sound [55, 60].

A deep neural network is essentially a parameterized function that decomposes into a hierarchical structure by design, and is trained *end-to-end* to learn a desired mapping, such as the tasks described above. End-to-end training implies that the optimization process operates on all parameters lying on the path from input to output. A stochastic optimization procedure called *stochastic gradient descent* [69] (SGD) is used over a *training set*, which is a collection of examples of true mappings. Hyperparameters for the model, such as design choices and optimization controls, are tuned using a held-out *validation set*. Final evaluation takes place upon an unseen *test set*. For deep networks, since the model possesses a hierarchical structure, applying SGD relies on application of the chain rule as we move down from the output layer to to lower layers, and this technique is called *backpropagation* [4]. In the following paragraphs, we shall describe a basic neural network architecture, the *feedforward neural network*, and a specific variety that has been shown to be exceptionally well-suited for handling image signals efficiently, the *convolutional neural network*.

Feedforward neural network An L-layer feedforward network is a parameterized function $f_{\boldsymbol{\theta}}$ that decomposes into L component functions that together compose f in order to learn a mapping from an input space $\mathcal{X} \in \mathcal{R}^d$ to an output space $\mathcal{Y} \in \mathcal{R}^m$:

$$f_{\boldsymbol{\theta}}(\boldsymbol{x}) = f^{(L)}_{\boldsymbol{\theta}_L}(f^{(L-1)}_{\boldsymbol{\theta}_{L-1}}(...f^{(1)}_{\boldsymbol{\theta}_1}(\boldsymbol{x})..)), \tag{1.12}$$

where the functions $f^{(i)}_{\boldsymbol{\theta}_i}$ typically take the form

$$f^{(i)}_{\boldsymbol{\theta}_i} = \sigma(\boldsymbol{W}_i \boldsymbol{h}_{i-1} + \boldsymbol{b}_i), \tag{1.13}$$

where σ is a non-linear *activation function*, $\boldsymbol{W}_i$ is a matrix of *weights*, $\boldsymbol{b}_i$ is a *bias* vector, and $\boldsymbol{h}_{i-1}$ is the *activation vector* from the previous layer (which is the result of applying the composition of

[4]See http://people.idsia.ch/~juergen/who-invented-backpropagation.html.

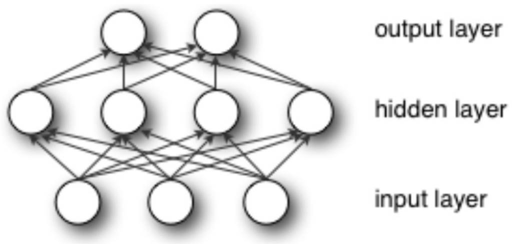


Figure 1.1: A feedforward network with one hidden layer; the computation of the hidden layer *activations* can be expressed as a matrix multiplication of a weight matrix with the input layer units, and similarly the output layer activations is a matrix multiplication of a different weight matrix with the hidden layer activations. Diagram taken from `http://deeplearning.net/tutorial/mlp.html`.

functions from $f^{(1)}$ to $f^{(i-1)}$).

For classification problems, the final layer usually outputs a vector the size of the number of classes and applies a *softmax* function to produce a vector of probabilities corresponding to the likelihood of the current input belonging to each of the possible classes:

$$p(y_j = 1|\boldsymbol{x}) = \frac{\exp(h_L^j)}{\sum_{i=1}^{m} \exp(h_L^i)}, \tag{1.14}$$

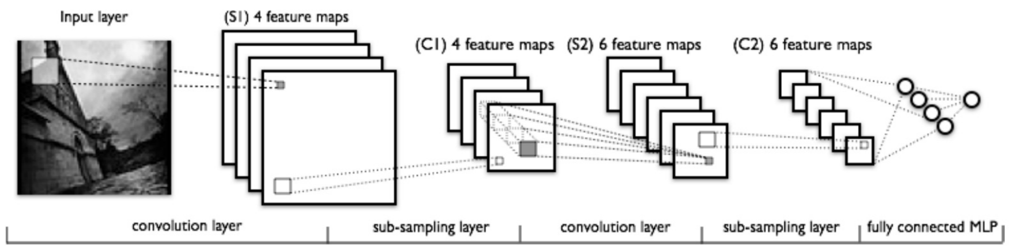
where m is the total number of classes, and typically the last layer does not apply a non-linearity. [5] The non-linearity σ in the previous layers is usually one of *sigmoid*, *tanh*, *Rectified Linear Unit* (ReLU), *Leaky ReLU*, with the most commonly used non-linearity nowadays being ReLU. These functions are defined below:

$$\begin{aligned}
\mathrm{sigmoid}(x) &= \frac{1}{1+\exp(-x)}; && (1.15)\\
\tanh(x) &= \frac{\exp(x)-\exp(-x)}{\exp(x)+\exp(-x)}; && (1.16)\\
\mathrm{ReLU}(x) &= x \text{ if } x \geq 0, \text{ and } 0 \text{ otherwise}; && (1.17)\\
\mathrm{LeakyReLU}(x) &= x \text{ if } x \geq 0, \text{ and } \alpha x \text{ otherwise } (\alpha \ll 1). && (1.18)
\end{aligned}$$

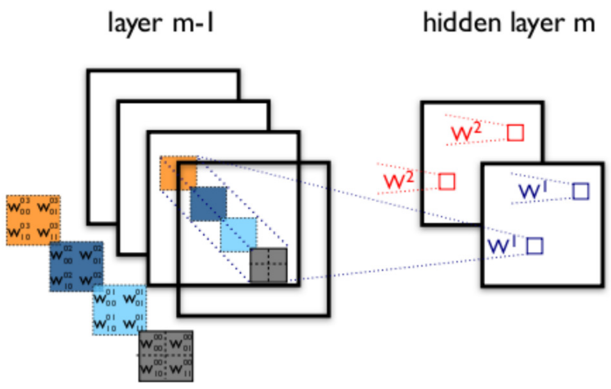
Convolutional neural network For data that posseses an intrinsic topology, imposing a corresponding structural prior on the weights $\boldsymbol{W}$ makes sense. For example, images obey a 2-D spatial topology, and sequential data typically share patterns across a temporal axis. This is the motivation behind *convolutional neural networks (CNNs)* [41]. The parameters of a convolutional layer consist of a set of *kernels* which is essentially a collection of small square matrices. These kernels are (discretely) convolved with the input to the layer and passed through a non-linearity, producing a set of *feature maps* (see Figure 1.2). Discrete convolution essentially implies flipping the kernel (or not, according to user choice), performing an elementwise multiplication of the kernel with the input values lying under it and summing these products to compute the output at the center position; and by sliding the kernel over all positions of the inputs and performing the same operation, the entire output is computed.

A convolutional neural network will typically maintain the spatial dimensions (height and width) of the input, and vary the number of channels (which is three at the input due to the three color channels, or one for grayscale images). Currently, the most widely adopted technique to downsample

[5] Without any non-linearities at all, the overall mapping would be effectively linear and little would be gained.



(a) A depiction of a classical convolutional network structure; nowadays, the sub-sampling layers are collapsed into the convolution layers by using strided convolutions (details in text).



(b) A feature map is created by summing convolutions with a filter across input channels.

Figure 1.2: Convolutional neural networks perform 2-D convolution of the input with kernels to share features across positions in the image. Diagrams taken from http://deeplearning.net/tutorial/lenet.html.

or upsample across the layers is a variant called *strided convolution*. Downsampling can be useful for scaling down to more compact "semantic spaces", and upsampling is desired in a convolutional generator network, for example. Discrete convolution of an image with a kernel involves the kernel sliding over the image one pixel at a time. Strided convolutions differ in how many pixels they shift by at every step – if they shift by two pixels instead of one, the resulting feature map will have spatial dimensions which are half that of the input. In this manner, strided convolutions can be used to downsample. For upsampling, the corresponding variant is *fractionally strided convolutions* (also called *transposed convolutions*, and previously *deconvolutions* [95]): one can think of a 2-D discrete convolution as a matrix multiplication of the image unrolled into a vector with a sparse matrix whose sets of rows consist of cyclic shifts of the rows of the kernel, and this result can be reshaped into feature maps of the correct height and width. For a strided convolution of stride 2, the cyclic shifts would be two steps at a time. From this it is easy to see that to scale up the spatial dimensions by 2 instead of scaling down, we can simply transpose the sparse matrix. This matrix would still have cyclic shifts corresponding to some kernel, but now the output shape has spatial dimensions that are double that of the input shape, which corresponds to the output and input shapes of the strided version [6].

Note that all of the above discussion generalizes to N-D convolutions; for instance, for 3-D (video data) we would have a cuboid kernel, and so on.

[6]There is a tutorial discussing this in more detail at http://deeplearning.net/software/theano/tutorial/conv_arithmetic.html.

Optimization (gradient descent/backpropagation) Let's say we are trying to fit a parameterized function $\boldsymbol{f_\theta}$ to a learn a mapping $\boldsymbol{f_\theta} : \mathcal{X} \rightarrow \mathcal{Y}$ given a dataset $\{(\boldsymbol{x}^{(1)}, \boldsymbol{y}^{(1)}), \cdots (\boldsymbol{x}^{(N)}, \boldsymbol{y}^{(N)})\}$ such that $(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)}) \in (\mathcal{X}, \mathcal{Y})$. Let's also assume we have defined a scalar *loss function* ℓ that quantifies how much we lose if our learned function's prediction $\boldsymbol{f_\theta}(\boldsymbol{x})$ does not match the ground truth $\boldsymbol{y}$, and we shall also assume that this loss function is differentiable in $\boldsymbol{\theta}$. Then we can learn a fit by performing the following update iteratively:

$$\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \alpha \nabla_{\boldsymbol{\theta}} \frac{1}{N} \sum_{i=1}^{N} \ell(f_{\boldsymbol{\theta}_{t-1}}(\boldsymbol{x}^{(i)}), \boldsymbol{y}^{(i)}), \tag{1.19}$$

where α is a hyperparameter controlling the stepsize of every update. This update rule is called *gradient descent* [26]. Note that to perform each update step, a pass through the entire dataset is required. Stochastic gradient descent [69] speeds up this process by performing an update after processing one example at a time, and this can be generalized to a minibatch of randomly selected examples for each update. A stochastic or minibatch gradient computation of this kind is an approximation to the true gradient, and this also adds noise/jitter to every update which can be useful when optimizing on unfriendly loss surfaces.

For the case of neural networks, $\boldsymbol{f_\theta}$ is a composition of L functions (Equation 1.12), so to compute gradients for the parameters in the l-th layer $\boldsymbol{\theta}_l$, we would need to use the chain rule to express the gradient of the loss *wrt* the parameters $\boldsymbol{\theta}_l$ in terms of the gradients of the loss *wrt* the activations in the layers $l+1$ to L ($\boldsymbol{h}_{l+1} \cdots \boldsymbol{h}_L$). Let's say we have a composition of 2 functions such that

$$\boldsymbol{h}_1 = f^{(1)}_{\boldsymbol{\theta}_1}(\boldsymbol{x}),$$

$$\hat{\boldsymbol{y}} = f^{(2)}_{\boldsymbol{\theta}_2}(\boldsymbol{h}_1)$$

and a specified loss function $\ell(\hat{\boldsymbol{y}}, \boldsymbol{y})$. To update the parameters in the first layer $\boldsymbol{\theta}_1$ we shall need the gradient of the scalar loss *wrt* $\boldsymbol{\theta}_1$. Using the chain rule from calculus [44, 45], we shall have

$$\nabla_{\boldsymbol{\theta}_1} \ell(\hat{\boldsymbol{y}}, \boldsymbol{y}) = \left(\frac{\partial \boldsymbol{h}_1}{\partial \boldsymbol{\theta}_1}\right)^T \nabla_{\boldsymbol{h}_1} \ell(\hat{\boldsymbol{y}}, \boldsymbol{y})). \tag{1.20}$$

Applying this rule recursively we can compute gradients for parameters in every preceeding layer by propagating the gradients of the losses *wrt* activations in the succeeding layers. This process is called *backpropagation* [87, 71], and forms the basis of training neural networks.

Modern neural networks use variants of SGD that use adaptive step sizes and statistics about past gradient steps to induce momentum-like dynamics to the optimization procedure [83, 35].

Regularization Deep neural networks are highly expressive and can *overfit* easily: this means the predictions will learn to map the given inputs to the given outputs very well, but will not perform well when presented with new unseen examples. Another way of looking at it is that the function will have a low *bias* (since a complex function has fewer simplifying assumptions), but learn very different functions (all of which learn the training set well) if trained on different sets, leading to a high *variance* in predictions on unseen data. To overcome this issue of learning overly complicated fits to data that do not generalize well, we use *regularization* techniques to discourage overfitting. Some of the prevalent techniques are penalizing the weights in the network from taking large values (intuitively, large weights can create more distorted functions which can discover more complex fits to data while small and sparse weights would create "simpler" functions that are less likely to overfit), *dropout* [78] (this technique zeros out weights randomly during training with the intuition of reducing co-adaptation of activations across layers; this process can also be interpreted as model averaging across weaker models, where the weakness is induced by setting a fraction of the weights to zero randomly), other tricks such as early stopping of training, pruning network weights, learning a lower capacity network, etc.

Batch normalization A technique that has been very useful in making training easier for neural networks is *batch normalization* [32]. As neural networks train, the distributions of the hidden activations at each layer keep changing (a problem refered to as *internal covariate shift*). Hypothesizing that reducing this shift would help optimization and generalization, Ioffe and Szegedy [32] propose to standardize the activations (prior to applying the non-linearity) using the sample mean and variance computed over the current minibatch. Doing this reduces the representation power of each layer, so additional parameters are learned for each layer that scale and shift the activations. [7] At test time, mean and variance parameters over the entire training set are used. In a sense, batch normalization adds an extra learnable dimension to training hierarchical models – that of learning the scale at each layer so that successive representations that build upon this lower layer are more useful.

Residual networks A recent architectural innovation that has enabled a significant jump in performance for deep learning is *residual networks* [27], which allow stable learning with high performance from networks with much deeper layers than previously trainable. This architecture won the top position in a number of competitions including image classification, object detection, and semantic segmentation tasks. [8] The key idea is to add a shortcut connection from the input to the current layer to a subsequent layer, which is an identity transform and is summed with the output of the current layer. This means that every layer learns a *residual* mapping and the network is free to learn a zero residue if newer layers are not beneficial. The underlying hypothesis is that it's easier to learn small residual mappings that can accumulate over many such mappings than learn it end-to-end through an equally deep "traditional" convolutional neural network. Additionally, gradients can propagate without vanishing due to the shortcut path.

1.5.3 VAE: Using deep neural networks for variational inference

Recall that variational autoencoders perform amortized variational inference, learning both model and variational parameters jointly via stochastic gradient descent. The variational posterior and the generative model are represented by deep convolutional neural networks, operating as the inference and generation machines, and both the variational and model parameters are learned jointly by stochastic gradient descent. A key variance reduction technique that is used is *stochastic backpropagation* which we describe below.

Stochastic backpropagation For computing gradients *wrt* parameters in models with continuous latent variables, the naïve method $\nabla_{\boldsymbol{\phi}} \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z})} \left[f(\boldsymbol{z}) \right] = \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z})} \left[f(\boldsymbol{z}) \nabla_{q_{\boldsymbol{\phi}}(\boldsymbol{z})} \log q_{\boldsymbol{\phi}}(\boldsymbol{z}) \right]$ when approximated with Monte-Carlo samples [9] is a high variance gradient estimator [61]. Stochastic backpropagation reparameterizes a random variable (in this case, $\boldsymbol{z}$) as a function of scale and shift parameters over a base distribution, and performs gradient-based learning *wrt* these parameters over Monte-Carlo samples drawn from the base distribution. This gradient estimator provides unbiased, low-variance gradients. As an example, if $q_{\boldsymbol{\phi}}(\boldsymbol{z})$ is a Normal distribution with mean and variance parameters $\boldsymbol{\phi} = \{\boldsymbol{\mu}, \boldsymbol{\sigma}\}$, then the reparameterization will use the $\mathcal{N}(\mathbf{0}, \boldsymbol{I})$ base distribution

[7] The shifting parameter can be absorbed into the bias term and so either may be dropped.

[8] See description in `https://github.com/KaimingHe/deep-residual-networks`.

[9] *Monte-Carlo* sampling is used to refer to repeated sampling of a stochastic process, and most often used (in this thesis) to approximate expectations with sums over the samples.

to express the random variable $\boldsymbol{z}$ as $\boldsymbol{z} = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon}; \boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$. Gradients of the expectation over $q_{\boldsymbol{\phi}}$ *wrt* the parameters $\boldsymbol{\phi}$ can be computed thus:

$$\nabla_{\phi} \underset{\boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})}{\mathbb{E}} \Big[f(\boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon}) \Big] = \underset{\boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})}{\mathbb{E}} \Big[\nabla_{\phi} f(\boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon}) \Big], \tag{1.21}$$

where $\phi \in \boldsymbol{\phi}$.

Variational autoencoder Neural networks are used to model the approximate inference and generator functions. The generator network, parameterized by $\boldsymbol{\theta}$, models a multivariate Gaussian in the case of real data, and Bernoulli for binary data. The inference network, parameterized by $\boldsymbol{\phi}$, is defined as a multivariate Gaussian with a diagonal covariance matrix $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) = \mathcal{N}(\boldsymbol{z}; \boldsymbol{\mu}(\boldsymbol{x}; \boldsymbol{\phi}), diag(\boldsymbol{\sigma}(\boldsymbol{x}; \boldsymbol{\phi})))$. The prior is chosen as the centered, isotropic Gaussian, $p_{\boldsymbol{\theta}}(\boldsymbol{z}) = \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$. The parameters $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ are learned by optimizing the variational lower bound we derived in the previous section, replacing the expectations over $q_{\boldsymbol{\phi}}$ with the reparameterization trick discussed above. The KL penalty term can be exactly computed between a Gaussian posterior and a Gaussian prior. The Monte-Carlo estimate for the lower bound is then given by:

$$\ell(\boldsymbol{\theta}, \boldsymbol{\phi}; \boldsymbol{x}) \approx \frac{1}{2} \sum_{j=1}^{J} \Big(1 + \log \boldsymbol{\sigma}(\boldsymbol{x}; \boldsymbol{\phi})_j^2 - \boldsymbol{\mu}(\boldsymbol{x}; \boldsymbol{\phi})_j^2 - \boldsymbol{\sigma}(\boldsymbol{x}; \boldsymbol{\phi})_j^2 \Big) + \frac{1}{L} \sum_{l=1}^{L} \log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}^{(l)}), \tag{1.22}$$

where $\boldsymbol{z}^{(l)} = \boldsymbol{\mu}(\boldsymbol{x}; \boldsymbol{\phi}) + \boldsymbol{\sigma}(\boldsymbol{x}; \boldsymbol{\phi}) \odot \boldsymbol{\epsilon}$ and $\boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$. J is the number of dimensions in the latent space, and L is the number of samples drawn per datapoint $\boldsymbol{x}$ (and in practice $L = 1$ works well enough).

For training via stochastic optimization, instead of the entire training set $\mathbf{X}$, minibatches are created with M randomly sampled datapoints, $\boldsymbol{X}^M = \{\boldsymbol{x}^{(i)}\}_{i=1}^{M}$, and these are used to approximate the bound per update:

$$\ell(\boldsymbol{\theta}, \boldsymbol{\phi}; \boldsymbol{X}) \approx \frac{N}{M} \sum_{\boldsymbol{x}^{(i)} \in \boldsymbol{X}^M} \tilde{\ell}(\boldsymbol{\theta}, \boldsymbol{\phi}; \boldsymbol{x}^{(i)}), \tag{1.23}$$

and now stochastic gradients may be computed for this loss function over the generator and inference parameters, and the entire model can be trained end-to-end using backpropagtion. Note that the gradients are actually *doubly* stochastic, because we approximate the gradient of the bound with minibatches, and also use a Monte-Carlo sampling in the expression for the bound.

1.6 PixelRNNs and PixelCNNs

In this section, we shall present a powerful class of implicit, fully-observed models. But first, we present background material on recurrent neural networks and a specific variant, the *long short-term memory*.

1.6.1 Recurrent neural networks (RNNs)

A simple RNN that outputs a prediction $\boldsymbol{y}^{(t)}$ at each time step, taking in an input $\boldsymbol{x}^{(t)}$ would update the hidden state $\boldsymbol{h}^{(t)}$ sequentially at every time step, by passing a sum of linear transformations of the previous hidden state $\boldsymbol{h}^{(t-1)}$ and the input $\boldsymbol{x}^{(t)}$ (along with bias terms) through a non-linearity, finally producing a prediction $\boldsymbol{y}^{(t)}$ by throwing a softmax (for a multi-class output) on top. In summary,

$$\boldsymbol{h}^{(t)} = \tanh(\boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x} + \boldsymbol{b}), \tag{1.24}$$

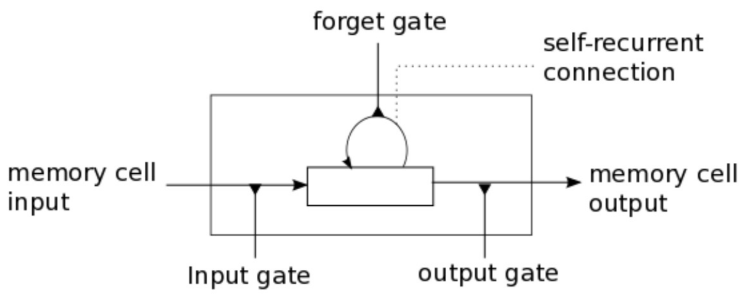


Figure 1.3: Memory cell in an LSTM. Diagram taken from http://deeplearning.net/tutorial/lstm.html.

$$\boldsymbol{y}^{(t)} = \operatorname{softmax}(\boldsymbol{V}\boldsymbol{h}^{(t)} + \boldsymbol{c}), \tag{1.25}$$

where $\boldsymbol{W}$, $\boldsymbol{U}$, and $\boldsymbol{V}$ are the the weight matrices corresponding to the hidden-to-hidden, input-to-hidden, and hidden-to-output connections respectively.

There are other variants, but this is the simplest and most representative. Depending on the specific task at hand (classifying an entire sequence, generating an entire sequence from a single input, reconstructing a sequence), one can come up with specific architectures that are variants of this.

Training an RNN takes place by unrolling the recurrent network, summing the losses at every time step, and computing the gradient of this total loss *wrt* the parameters of the network $\{\boldsymbol{W}, \boldsymbol{U}, \boldsymbol{b}, \boldsymbol{V}, \boldsymbol{c}\}$ of the simple model described above. This process is referred to as *Back-Propagation-Through-Time* (BPTT) [88].

A number of tricks are typically used for training RNNs, some of which are as follows.

- *teacher forcing*: When the prediction at intermediate time steps depends on predictions at previous time steps, we feed the actual ground truth from previous predictions into the predictor for the current time step at training time.
- *truncated backpropagation*: We unroll the network for only a short number of steps backward in time, and not all the way to the start.
- *gradient clipping*: We clip the gradients to have bounded norm, which helps avoid gradient divergence.
- *clever initialization*: We can initialize weights to be a scaled identity matrix, or even restrict the parameter space so that the (hidden-to-hidden) weight matrices are orthogonal (or unitary as in Arjovsky, Shah, and Bengio [4]) so that gradients don't diverge over long sequences.

1.6.2 Long short-term memory (LSTM)

LSTMs [30] are a variant of RNNs, where in addition to the hidden state $\boldsymbol{h}^{(t)}$, we also have a *cell state* $\boldsymbol{c}^{(t)}$ that is maintained via an internal recurrence (or self-loop), and the intended purpose of this node is to maintain memory of events over a long term. This cell state is updated at every time step by "erasing" values in selected positions in the cell state by elementwise multiplication with a *forget gate* $\boldsymbol{f}^{(t)}$, and summing new elements, $\tilde{\boldsymbol{c}}^{(t)}$ with an "importance" scaled by an *input gate* $\boldsymbol{i}^{(t)}$

(see Figure 1.3). An *output gate* $\boldsymbol{o}^{(t)}$ acts as a gating mechanism for computing the hidden state as a function of the cell state. The computation of these vectors is as follows:

$$\begin{aligned}\boldsymbol{f}^{(t)} &= \sigma(\boldsymbol{W}_f.[\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}] + \boldsymbol{b}_f), &(1.26)\\ \boldsymbol{i}^{(t)} &= \sigma(\boldsymbol{W}_i.[\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}] + \boldsymbol{b}_i), &(1.27)\\ \boldsymbol{o}^{(t)} &= \sigma(\boldsymbol{W}_o.[\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t}] + \boldsymbol{b}_o), &(1.28)\\ \tilde{\boldsymbol{c}}^{(t)} &= \tanh(\boldsymbol{W}_c.[\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}] + \boldsymbol{b}_c). &(1.29)\end{aligned}$$

The update equations for the cell state and hidden state are summarized below:

$$\begin{aligned}\boldsymbol{c}^{(t)} &= \boldsymbol{f}^{(t)} \odot \boldsymbol{c}^{(t-1)} + \boldsymbol{i}^{(t)} \odot \tilde{\boldsymbol{c}}^{(t)}, &(1.30)\\ \boldsymbol{h}^{(t)} &= \boldsymbol{o}^{(t)} \odot \tanh(\boldsymbol{c}^{(t)}). &(1.31)\end{aligned}$$

Variants on the above computations exist, such as the Gated Recurrent Unit (GRU) [14, 15], which uses a single gate and its complement for both forgetting and updating, and a single state that acts as a hidden state that is designed to hold both short term as well as long term memory.

LSTMs are currently the best performing recurrent models when it comes to sequences, and the term is almost synonymous with RNNs nowadays.

1.6.3 PixelRNN and PixelCNN

One can tractably model a joint distribution over a set of n random variables, $\boldsymbol{x} = \{x_1, \cdots, x_n\}$, by using the chain rule of probability and decomposing the joint as a product of conditional distributions:

$$p(x_1, \cdots, x_n) = p(x_1)p(x_2|x_1)p(x_3|x_2, x_1) \cdots p(x_n|x_{n-1} \cdots x_1). \tag{1.32}$$

Oord, Kalchbrenner, and Kavukcuoglu [57] introduced PixelRNN, where they use LSTM network layers to model the conditional distributions of every pixel conditioned on the previous ones. The ordinal notion of "previous" is based on an assigned ordering of pixels, which in this case is rows of pixels from top to bottom, with pixels in each row ordered left to right. In addition, another important modification is the modelling of pixels as discrete multinomial distributions rather than continuous valued random variables. This was observed to be an easier output distribution to optimize over than a continuous mixture of densities, as was the standard practice. Training is parallel since the ground truth pixel values can be used at training time (teacher forcing), but generation is sequential because the distribution for each pixel is conditioned on previous pixel predictions. The work introduced three architectural variants: *Row LSTM*, *Diagonal BiLSTM*, and *PixelCNN*, which are briefly described next.

Row LSTMs This variant uses convolutions to compute features for every pixel in the image. The convolutions are *masked* (so that the contribution of pixels to the right of, and including the center is zero) in order to maintain the chain of dependency from left to right. The LSTM itself predicts an entire row of pixels at every time step, which maintains the chain of dependency from top to bottom. Together, for modelling the output distribution of every pixel, the hidden state from the previous LSTM step carries over (partial) information from all the previous rows, while the current (masked) row convolution provides (partial) information about the pixels in the current row predicted so far. (The information is partial because it's dependent on the receptive field of the convolutions.) The update equations for the i-th row of pixels in the image are as follows:

$$\begin{aligned}[\boldsymbol{o}^{(i)}, \boldsymbol{f}^{(i)}, \boldsymbol{i}^{(i)}, \tilde{\boldsymbol{c}}^{(i)}] &= \text{activation}(\boldsymbol{K^{hh}} * \boldsymbol{h}^{(i-1)} + \boldsymbol{K^{ih}} * \boldsymbol{x}^{(i)}), &(1.33)\\ \boldsymbol{c}^{(i)} &= \boldsymbol{f}^{(i)} \odot \boldsymbol{c}^{(i-1)} + \boldsymbol{i}^{(i)} \odot \tilde{\boldsymbol{c}}^{(i)}, &(1.34)\end{aligned}$$

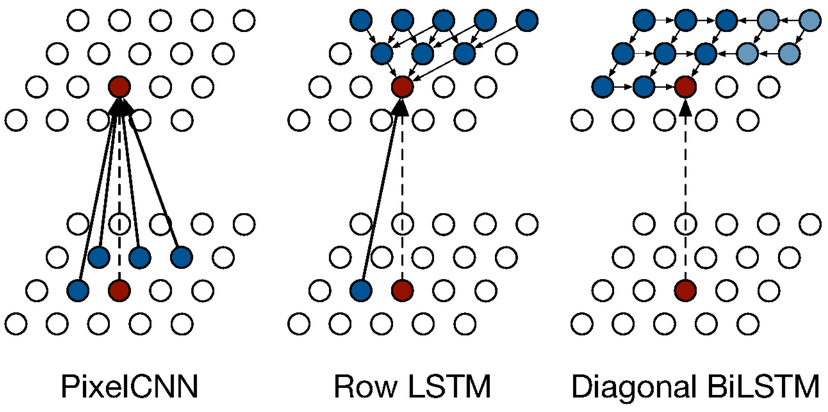


Figure 1.4: Receptive fields for PixelCNN *(left)*, Row LSTM *(center)*, and Diagonal BiLSTM *(right)*. The arrows denote local context for a position. Walking backwards along a directed path gives us the global context. Reproduced with permission from Oord, Kalchbrenner, and Kavukcuoglu [57].

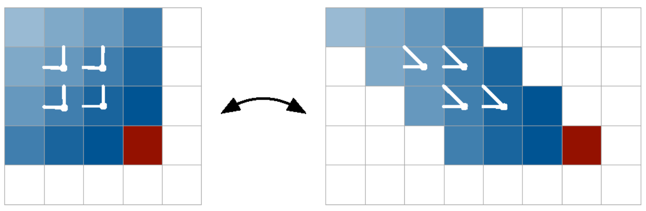


Figure 1.5: In the Diagonal BiLSTM, skewing the image results in being able to apply efficient 2×1 columnwise convolutions which conditions upon pixels to the top and right of the center pixel. Reproduced with permission from Oord, Kalchbrenner, and Kavukcuoglu [57].

$$\boldsymbol{h}^{(i)} \quad = \quad \boldsymbol{o}^{(i)} \odot \tanh(\boldsymbol{c}^{(i)}), \tag{1.35}$$

where $*$ is a convolution operator, $\odot$ is the Hadamard operator; *activation* is the *sigmoid* function for the output, forget, and input gates, and *tanh* for the cell state; $\boldsymbol{K^{hh}}$ is the 1-D convolution kernel for the hidden-to-hidden component, and $\boldsymbol{K^{ih}}$ is the 1-D masked kernel for the input-to-hidden component. These two components, when summed, are of size $(4 \times \#channels \times height \times width)$, and capture a context to the left of the center pixel (of size $\frac{side(\boldsymbol{K^{ih}})-1}{2}$) and a triangular shaped context from the above rows as induced by the receptive field of the kernel $\boldsymbol{K^{hh}}$.

Diagonal BiLSTM This variant solves the problem of not having the full context to condition the predictive distribution of the pixel upon, which the Row LSTM suffers from. The trick is to perform 2×1 convolutions along diagonals so that every pixel prediction is conditioned upon the pixel on top and the one to the left. Maintaining this receptive field throughout gives every pixel a receptive field that covers all of the pixels to the left and top (dark blue nodes in Figure 1.4). For a full context we also need the nodes labeled light blue, so another component works on a horizontally flipped version of the image and performs the same operation. In order to not use the pixels to the right of the center pixel, this flipped version is shifted one row down. The other trick to perform the diagonal convolution efficiently and in parallel is to skew the image so that diagonals in the image correspond to columns in the skewed image: now column-wise convolutions on the skewed image correspond to diagonal convolutions on the original image (see Figure 1.5). The hidden states computed by the two LSTMs (corresponding to the skewed and the flipped-shifted-skewed images) are summed to achieve the full context for a pixel. The LSTMs use the same computations as in Equation 1.35 with the following differences: the LSTM states are now predictive of image diagonals at every step instead of rows, $\boldsymbol{K^{hh}}$ is a 2×1 convolution kernel operating column-wise (since a larger size does not give us any more useful context), $\boldsymbol{K^{ih}}$ is a convolution kernel of size 1×1 that uses information at the center location in higher layers of the LSTM, except the first, where it contributes zero information since we don't want to condition on the variable we are modelling.

PixelCNN Modelling full contexts with recurrent network layers is computationally expensive, so the PixelCNN simplifies this requirement by modelling a context covered by the receptive field of a convolutional neural network, with the modification that the convolutional filters are masked to zero out any contribution from pixels to the right and bottom of the center pixel. The context available is the region corresponding to roughly half the filter size as shown in Figure 1.4. Stacking convolutional layers increases this receptive field by half the filter size per layer.

Some other implementation details:

- All convolutional filters used have two types: A and B. Filters of type A do not allow contribution from the center pixel, so these are used in the first layers; filters of type B allow it, and are used in subsequent higher layers (since this actually conditions on the center prediction in the previous layer and not the pixel value we are modelling). Additionally, for 3-channel colour images, the pixel values for the Blue and Green channels are allowed to use the networks predictions for the Red and (Red, Blue) values respectively by mask B.

- Multiple levels of LSTM layers are used for PixelRNN, skip connections are used between layers, and residual blocks are used for the multilayer PixelCNN (the number of layers should ideally be enough to span a receptive field the size of the image).

Gated PixelCNN and blind spot fix In follow up work Oord et al. [59] improve upon the PixelCNN by making a few changes to the architecture:

- The convolution layers of PixelCNN are modified to include a gating mechanism, which is expected to put the capacity of the network on par with a better performing LSTM-based PixelRNN. The way this is done is by splitting up the feature maps into two parts and using a *sigmoid* transform of one half to act as a multiplicative gating unit for a *tanh* transform of the other half.

- A CNN with masked filters as used by the basic PixelCNN described above has a receptive field with a growing blind spot owing to the cumulative effect of the masking as one reconstructs the receptive field induced by the masked filters over the previous rows. To provide the full context, a modification was proposed that splits up the conditional model for a pixel into two components - one uses a set of filters that have their entire lower half masked thus inducing a full receptive field over all the previous rows, and the other component is a masked 1-D convolution *à la* Row LSTM, which operates on the current row to encode conditioning information about the pixels in the current row so far. These two components are combined to produce final predictive conditional distributions over the pixels.

This work also extends this class of density estimators into conditional generation models – it is shown that by adding conditioning information through a linear transform of a one-hot class-label vector, we can generate samples of specific classes.

1.7 Generative adversarial networks

Goodfellow et al. [23] introduced a generative model that trains a generator network G by pitting a discriminator network D against it. The discriminator network is trained to discriminate between samples from the true data distribution $\mathbb{P}_r(\boldsymbol{x})$ and those from the generator distribution $\mathbb{P}_g(\boldsymbol{x})$. The generator network is trained to produce samples that would fool the discriminator into thinking that its samples come from $\mathbb{P}_r(\boldsymbol{x})$.

Formally, the minimax objective is written as follows:

$$\min_G \max_D \mathop{\mathbb{E}}_{\boldsymbol{x}\sim\mathbb{P}_r}[\log(D(\boldsymbol{x}))] + \mathop{\mathbb{E}}_{\tilde{\boldsymbol{x}}\sim\mathbb{P}_g}[\log(1 - D(\tilde{\boldsymbol{x}}))], \tag{1.36}$$

D and G are typically convolutional neural networks for modelling natural images. $\mathbb{P}_g$ is the generator distribution implicitly defined by deterministically transforming a sample $\boldsymbol{z}$ from a simpler distribution $p(\boldsymbol{z})$ through the highly non-linear function G.

$$\boldsymbol{z} \sim p(\boldsymbol{z});\;\; \tilde{\boldsymbol{x}} = G(\boldsymbol{z}). \tag{1.37}$$

At the start of training the discrimininator is able to tell with high confidence if samples come from the generator distribution which leads the sigmoid classifier to saturate and produce very small gradients; this is multiplied by $1/(1 - D(G(\boldsymbol{z})))$, and this term is close to 1, so the overall gradient stays close to zero. Goodfellow et al. [23] fix this with a heuristic: they change the objective for the generator to a maximization problem rather than a minimization so that the pre-multiplier is now $1/D(G(\boldsymbol{z}))$, and this enhances the small gradient when $D(G(\boldsymbol{z}))$ goes to zero. The new objective is

$$\begin{aligned} &\max_D \mathop{\mathbb{E}}_{\boldsymbol{x}\sim\mathbb{P}_r}[\log(D(\boldsymbol{x}))] + \mathop{\mathbb{E}}_{\tilde{\boldsymbol{x}}\sim\mathbb{P}_g}[\log(1 - D(\tilde{\boldsymbol{x}}))], \\ &\max_G \mathop{\mathbb{E}}_{\tilde{\boldsymbol{x}}\sim\mathbb{P}_g}[\log(D(\tilde{\boldsymbol{x}}))]. \end{aligned} \tag{1.38}$$

Jensen-Shannon divergence The Jensen-Shannon divergence (JSD) is a symmetric divergence between two distributions p and q and is expressed as the average of the KL divergence between p and $\frac{p+q}{2}$ and that between q and $\frac{p+q}{2}$:

$$JSD(p, q) = \frac{1}{2}\left(KL(p \,||\, \frac{p+q}{2}) + KL(q \,||\, \frac{p+q}{2})\right). \tag{1.39}$$

Goodfellow et al. [23] show that in the presence of an optimal discriminator, *i.e.* one that models $\mathbb{P}_g(\boldsymbol{x})/(\mathbb{P}_r(\boldsymbol{x}) + \mathbb{P}_g(\boldsymbol{x}))$, the cost function can be seen as equivalently minimizing a Jensen-Shannon divergence between real and generated data.

1.7.1 Training GANs

GAN training is generally unstable, and there are a number of heuristics that are known to help. Some of the following are particularly useful, recommended by Chintala et al. [13]:

- Normalize inputs to lie between [-1, 1], and use a *tanh* at the last layer of the generator.
- Use a unit normal distribution to sample $p(\boldsymbol{z})$.
- The DCGAN architecture from Radford, Metz, and Chintala [64] is reliable.
- Sparse gradients contribute to instability, so it's advisable to use Leaky ReLUs and substitute max pooling layers with strided convolutions/transposed convolutions.
- The Adam optimizer [35] usually works best, and a recommended setting for a DCGAN-like architecture is α = 2e-4, $\beta_1 = 0.5, \beta_2 = 0.999$.

- Adding noise to inputs for the discriminator helps (this is discussed in more detail in the following section).

In addition, Salimans et al. [73] propose additional strategies for training GANs, specifically:

- *feature matching*, which asks the generator to match statistics of real data features at an intermediate layer to those of the corresponding layer in the discriminator,
- *minibatch discrimination*, which lets the discriminator use side-information from the other samples in a minibatch to classify the samples as real or generated (with an effect similar to that of batch normalization) and this encourages the generator to be diverse, thus reducing *mode collapse*,
- *historical averaging*, which discourages parameter drift away from a historical average of past values,
- *one-sided label smoothing*, which replaces the labels for real data with a soft target (0.9 instead of 1), and
- *virtual batch normalization*, which normalizes a minibatch using statistics from a separate reference minibatch (but this is an expensive technique because it requires an additional forward propagation on the reference batch).

Although these heuristics exist, training GANs (in the formulation described so far) is still known to be painful. A particular problem is that of *mode collapse*, where the generator learns that there are specific "sub-distributions" it can produce samples from that are more likely to fool the discrimininator. It then concentrates its probability density in these regions, completely dropping the other modes of the distribution it's modelling.

Wasserstein GAN (WGAN) from Arjovsky, Chintala, and Bottou [3] is a recent development that goes a long way toward identifying some of the issues with GAN training and stabilizing it.

1.7.2 Wasserstein GAN

Arjovsky and Bottou [2] study the problem of instability with GAN training. They show that for two distributions that have disjoint supports or lie on low dimensional manifolds, a perfect discriminator exists that has a zero gradient (almost everywhere) with respect to its input. There is evidence [54] that structured high dimensional data is indeed concentrated on a low dimensional manifold, and Arjovsky and Bottou [2] show that the generator distribution also lies on a low dimensional manifold, so a perfect discrimininator (under which the JSD minimization interpretation holds) would actually not provide a good training signal for a generator. In addition, it is shown that the $-\log D(G(\boldsymbol{z}))$ version of the cost function which avoids vanishing gradients leads to highly unstable gradient updates (in the sense of possessing a high variance). Adding continuous noise to $\mathbb{P}_r$ and $\mathbb{P}_g$ would extend the distributions to having overlapping supports, and now the JSD would not reach its optimum, but might be deceptively smaller due to the overlap. A training setup of annealing the noise while still training the discriminator until optimality is pointed out as a possibility, although the choice of variance and annealing schedule are still potentially finicky. Additionally, it is pointed out that while the JSD between $\mathbb{P}_r$ and the noisy $\mathbb{P}_r$ is always maxed out irregardless of annealing (since $\mathbb{P}_r$ always lies on a low-dimensional manifold), the *Wasserstein distance* between $\mathbb{P}_r$ and the noisy $\mathbb{P}_r$ is shown to have the desirable property of going smoothly to 0 as the noise is annealed, which hints at this being a more desirable choice of distance. In follow-up work, Arjovsky, Chintala, and Bottou [3] present a simple GAN training alternative, based on approximating the

Wasserstein distance. Next, we briefly describe this distance, and the Kantorovich-Rubinsten duality which forms the foundation for WGANs.

Wasserstein/Earth-Mover's distance The Wasserstein distance, also refered to as the *Earth-Mover's Distance* (EM distance) can be intuitively thought of as the minimum amount of effort required to move mass distributed according to one distribution to match another distribution (where the effort to move some mass is directly proportional to both the amount of mass to be moved and distance it needs to be moved).

Formally, for two distributions $\mathbb{P}_r$ and $\mathbb{P}_g$, and Π, the set of all joint distributions such that the marginal distributions of $\gamma \in \Pi(\mathbb{P}_r, \mathbb{P}_g)$ are $\mathbb{P}_r$ and $\mathbb{P}_g$, and a choice of distance metric in the transport space $d(\cdot, \cdot)$, the Wasserstein distance is expressed as

$$W_p(\mathbb{P}_r, \mathbb{P}_g) = \inf_{\gamma \in \Pi(\mathbb{P}_r, \mathbb{P}_g)} \left(\mathop{\mathbb{E}}_{(\boldsymbol{x}, \boldsymbol{y}) \sim \gamma} \left[d(\boldsymbol{x}, \boldsymbol{y})^p \right] \right)^{1/p} ; (p \geq 1). \tag{1.40}$$

This definition can be understood intuitively by seeing that the total mass leaving point $\boldsymbol{x}$ must be $\mathbb{P}_r(\boldsymbol{x})$, and the total mass entering point $\boldsymbol{y}$ must equal $\mathbb{P}_g(\boldsymbol{y})$, which is why the marginal condition must be satisfied. The total effort spent is then $\int_{\boldsymbol{x}} \int_{\boldsymbol{y}} \gamma(\boldsymbol{x}, \boldsymbol{y}) d(\boldsymbol{x}, \boldsymbol{y}) d\boldsymbol{y} d\boldsymbol{x}$ which is exactly the definition above for $p = 1$, which is what we shall henceforth use, along with the Euclidean norm as the distance measure (and this is the version that shall be implied when we say EM distance or W):

$$W(\mathbb{P}_r, \mathbb{P}_g) = \inf_{\gamma \in \Pi(\mathbb{P}_r, \mathbb{P}_g)} \mathop{\mathbb{E}}_{(\boldsymbol{x}, \boldsymbol{y}) \sim \gamma} \left[\, || \, \boldsymbol{x} - \boldsymbol{y} \, || \, \right]. \tag{1.41}$$

Arjovsky, Chintala, and Bottou [3] show that compared to other prevalent distances (such as the Kullback-Leibler, reverse Kullback-Leibler, and Jensen-Shannon divergence, or the Total Variation distance) the EM distance is continuous and differentiable almost everywhere (under mild assumptions), which makes it a better choice for a loss function corresponding to a distance between distributions that lie in low-dimensional manifolds. It is also shown that a small EM distance corresponds to a small difference in distributions, and that any distribution $\mathbb{P}_g$ that converges under the other divergences and distances [10] also converges under the EM distance.

Kantorovich-Rubinstein duality Unfortunately, the EM distance is highly intractable; from its expression in Equation 1.40 it can be seen that the infinimum operates over a pretty large set of joint distributions coupled with computing an expected distance over every one of these distributions. Fortunately, the Kantorovich-Rubinstein duality [84] presents a dual formulation for the EM distance:

$$W(\mathbb{P}_r, \mathbb{P}_g) = \sup_{||D||_L \leq 1} \left(\mathop{\mathbb{E}}_{\boldsymbol{x} \sim \mathbb{P}_r} \left[D(\boldsymbol{x}) \right] - \mathop{\mathbb{E}}_{\boldsymbol{x} \sim \mathbb{P}_g} \left[D(\boldsymbol{x}) \right] \right), \tag{1.42}$$

where the condition under the supremum indicates that it is over all 1-Lipschitz functions $D : \mathcal{X} \to \mathcal{R}$. A function $D : X \to Y$ is K-Lipschitz if

$$d_Y(D(\boldsymbol{x}_1), D(\boldsymbol{x}_2)) \leq K d_X(\boldsymbol{x}_1, \boldsymbol{x}_2), \tag{1.43}$$

for distance functions d_X and d_Y defined on the spaces X and Y.

If we instead optimize over all K-Lipschitz functions, we would be optimizing K times the EM distance, which would also lead us to the same optima (albeit losing comparability of the distance

[10]This means that given a sequence of $\{\mathbb{P}_g^t\}_{t \in \mathcal{N}}$, the sequence converges to a specific distribution, $\mathbb{P}_g^\infty$.

across different function classes unless K is known). We shall consider a parameterized function D, which we shall think of as a critic to a generator. [11] If the generator G is parameterized by $\boldsymbol{\theta}$, we shall solve the following problem to find the optimal critic:

$$\max_{D \in \mathcal{D}_K} \left[\mathop{\mathbb{E}}_{\boldsymbol{x} \sim \mathbb{P}_r} \big[D(\boldsymbol{x})\big] - \mathop{\mathbb{E}}_{\boldsymbol{z} \sim p(\boldsymbol{z})} \big[D(G_{\boldsymbol{\theta}}(\boldsymbol{z}))\big] \right], \tag{1.44}$$

where $\mathcal{D}_K$ is the set of all K-Lipschitz functions. The generator can then be trained by computing the gradient of the distance *wrt* the parameters $\boldsymbol{\theta}$,

$$\nabla_{\boldsymbol{\theta}} W(\mathbb{P}_r, \mathbb{P}_g) = - \mathop{\mathbb{E}}_{\boldsymbol{z} \sim p(\boldsymbol{z})} \big[\nabla_{\boldsymbol{\theta}} D(G_{\boldsymbol{\theta}}(\boldsymbol{z}))\big]. \tag{1.45}$$

In order to use these two (alternating) update rules for a GAN where D and G are neural networks, we first need to impose the K-Lipschitz constraint on the set of functions $\mathcal{D}$: this can be done most simply by restricting the parameter space of $\mathcal{D}$ to a specified interval, say, $[-c, c]$. Since the (differentiable) function D has bounded and closed support, clipping the parameters would lead D to be bounded, and the bound would depend only on the clipping value and architecture [12]. Since $D(\boldsymbol{x})$ and $\boldsymbol{x}$ are both bounded, so is $d_Y(D(\boldsymbol{x}_1), D(\boldsymbol{x}_2))$ which implies D is K-Lipschitz for some K, as per the definition in Equation 1.43.

Advantages WGAN training is shown to be more stable than traditional GAN training. One can train a critic to convergence and there is no need to balance generator/discriminator updates; it has a loss function that appears to correlate well with sample quality and convergence of the generator; there is improved robustness in the sense that a range of architecture choices do not affect training or sample quality adversely; and there is qualitative evidence of reduced mode-collapse.

1.8 Brief summary of work presented in this thesis

In this thesis we will present three contributions involving generative models for natural images.

PixelVAE: Combining VAEs with PixelCNNs Recall that PixelCNN is a fully observed, explicit model, tractably modelling a joint distribution over the output. It is observed to produce samples with more precise local structure but which are somewhat lacking in global structure. Additionally, PixelCNN does not provide a latent representation that could potentially be of use in downstream tasks. Also recall that the VAE is a powerful latent variable model, assuming conditional independence among output dimensions. Unfortunately, VAEs are known to suffer from samples with poor local structure (but decent global structure) due to maximum likelihood training leading to a “blurring” effect due to hedging bets. In chapter 2, we show that a VAE with a PixelCNN decoder produces samples that have good local as well as global structure. The overall model is less costly than a pure PixelCNN, since we free the decoder from having to model global structure. Finally, we extend the model to a multi-level latent space and qualitatively demonstrate the decomposition of the latent representation into a hierarchy of semantic features.

Improved WGAN training Recall that Wasserstein GANs require the critic function to be constrained such that it is K-Lipschitz for some K, and Arjovsky, Chintala, and Bottou [3] implement this constraint by clipping the parameters of the neural network critic. In this work, we

[11]The term *critic* is used instead of *discriminator*because it gives us an unconstrained real number corresponding to the difference in distributions, as opposed to a sigmoid output corresponding to a classification into real/fake.

[12]For example, the number of layers dictates how many times and how the parameters get to interact with each other.

show that this leads to underuse of capacity, exploding and vanishing gradients in deeper networks, and optimization difficulties with specific architectures. In chapter 3, we present an alternative to encourage the Lipschitz property, based on penalizing a norm on the critic gradient *wrt* the input, and find that this variant performs better than parameter clipping and also enables stable training of a wide range of architectures, additionally working on character-level language models (in a discrete data space) without use of any heuristics that traditional GANs require to perform well on language generation.

Exploring the use of generative models as normality models of data Finally, in chapter 4, we explore the potential use of generative models as models of normality given a training distribution. This would allow us to perform tasks such as out-of-distribution detection, which would find application in quite a few areas that involve detecting novel and unexpected instances, and one motivating example for us is *AI safety* [13], since we would like our AI systems to know when they encounter a situation that is less likely under the data distribution they've been training on. Upon detecting such an instance successfully, they can raise a flag and ask for help, or stop acting, or whichever course of action is deemed safest by design. We evaluate performance of the VAE, PixelCNN, PixelVAE under a framework in which the model is trained on all but a subset of data (corresponding to a class label) and we compare likelihoods for in-distribution and out-of-distribution test data. We also evaluate DCGAN, by using the discriminator prediction for real/fake data as a surrogate for likelihood.

[13]We would like AI systems deployed in the real world to behave safely with regard to human interests.

Prologue to the first article

Title: *PixelVAE: A latent variable model for natural images.*
Authors: Ishaan Gulrajani, Kundan Kumar, **Faruk Ahmed**, Adrien Ali Taiga, Francesco Visin, David Vazquez, Aaron Courville.

Abstract *Natural image modeling is a landmark challenge of unsupervised learning. Variational Autoencoders (VAEs) learn a useful latent representation and model global structure well but have difficulty capturing small details. PixelCNN models details very well, but lacks a latent code and is difficult to scale for capturing large structures. We present PixelVAE, a VAE model with an autoregressive decoder based on PixelCNN. Our model requires very few expensive autoregressive layers compared to PixelCNN and learns latent codes that are more compressed than a standard VAE while still capturing most non-trivial structure. Finally, we extend our model to a hierarchy of latent variables at different scales. Our model achieves state-of-the-art performance on binarized MNIST, competitive performance on* 64×64 *ImageNet, and high-quality samples on the LSUN bedrooms dataset.*

This article was accepted for publication at the International Conference of Learning Representations, 2017.

The project was led by Ishaan Gulrajani, who wrote most of the code-base that the rest of us built upon. My personal contributions were:

- helping experiment with regularizers, optimizers, architecture sizes, normalizers, activation functions, and general hyper-parameters for stabilizing the base model in development phase,
- performing exploratory visualizations with latent codes for MNIST and LSUN bedrooms,
- helping with running some of the experiments with larger images (64×64 bedrooms and Imagenet-64),
- implementing and experimenting with sampling at different levels of the latent space. Also implementing an efficient sampler for PixelCNN in TensorFlow for sampling larger images (following discussions with Kundan Kumar and Ishaan Gulrajani, who jointly developed the precise idea for the implementation),
- contributing significantly to the writing and presentation of the paper.

Chapter 2

PixelVAE: A latent variable model for natural images

2.1 Introduction

Building high-quality generative models of natural images has been a long standing challenge. Although recent work has made significant progress [37, 57, 59], we are still far from generating convincing, high-resolution natural images.

Many recent approaches to this problem are based on an efficient method for performing amortized, approximate inference in continuous stochastic latent variables: the variational autoencoder (VAE) from Kingma and Welling [37] jointly trains a top-down "decoder" generative neural network with a bottom-up "encoder" inference network. VAEs for images typically use rigid decoders that model the output pixels as conditionally independent given the latent variables. The resulting model learns a useful latent representation of the data and effectively models global structure in images, but has difficulty capturing small-scale features such as textures and sharp edges due to the conditional independence of the output pixels, which significantly hurts both log-likelihood and quality of generated samples compared to other models.

PixelCNNs [57, 59] are another state-of-the-art image model. Unlike VAEs, PixelCNNs model image densities autoregressively, pixel-by-pixel. This allows it to capture fine details in images, as features such as edges can be precisely aligned. By leveraging carefully constructed masked convolutions [59], PixelCNNs can be trained efficiently in parallel on GPUs. Nonetheless, PixelCNN models are still very computationally expensive. Unlike typical convolutional architectures they do not apply downsampling between layers, which means that each layer is computationally expensive and that the depth of a PixelCNN must grow linearly with the size of the images in order for it to capture dependencies between far-away pixels. PixelCNNs also do not explicitly learn a latent representation of the data, which can be useful for downstream tasks such as semi-supervised learning.

Our contributions are as follows:

- We present PixelVAE, a latent variable model which combines the largely complementary advantages of VAEs and PixelCNNs by using PixelCNN-based masked convolutions in the conditional output distribution of a VAE.
- We extend PixelVAE to a hierarchical model with multiple stochastic layers and autoregressive decoders at each layer. This lets us autoregressively model not only the output pixels but also higher-level latent feature maps.
- On MNIST, we show that PixelVAE: (1) establishes a new state-of-the-art likelihood, (2) performs comparably to PixelCNN using far fewer computationally expensive autoregressive

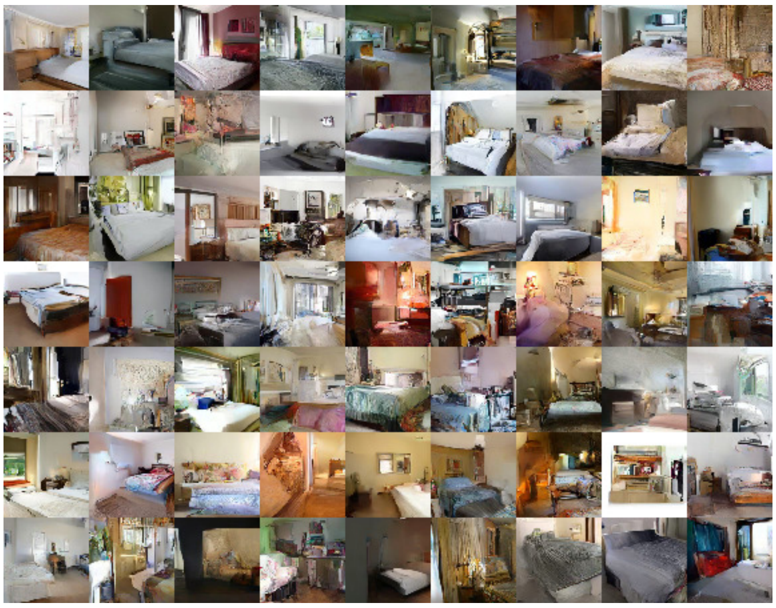


Figure 2.1: Samples from hierarchical PixelVAE on the LSUN bedrooms dataset.

layers, (3) learns more compressed latent codes than a standard VAE while still accounting for most non-trivial structure, and (4) learns a latent code which separates digits better than a standard VAE.

- We evaluate hierarchical PixelVAE on two challenging natural image datasets (64×64 ImageNet and LSUN bedrooms). On 64×64 ImageNet, we report likelihood competitive with the state of the art at significantly less computational cost. On LSUN bedrooms, we generate high-quality samples and show that hierarchical PixelVAE learns to model different properties of the scene with each of its multiple layers.

2.2 Related work

There have been many recent advancements in generative modelling of images. We briefly discuss some of these below, especially those that are related to our approach.

The *variational autoencoder* (VAE) [37] is a framework to train neural networks for generation and approximate inference jointly by optimizing a variational bound on the data log-likelihood. The use of normalizing flows [67] improves the flexibility of the VAE approximate posterior. Based on this, [36] develop an efficient formulation of an autoregressive approximate posterior model using MADE [22]. In our work, we avoid the need for such flexible inference models by using autoregressive priors.

The idea of using autoregressive conditional likelihoods in VAEs has been explored in the context of language modelling in [7], however in that work the use of latent variables fails to improve likelihood over a purely autoregressive model.

Simultaneously to our work, Chen et al. [12] present a VAE model for images with an an autoregressive output distribution. In constrast to Chen et al. [12], who focus on models with a single layer of latent variables, we also investigate models with a hierarchy of latent variables (and corresponding autoregressive priors) and show that they enable us to scale our model to challenging

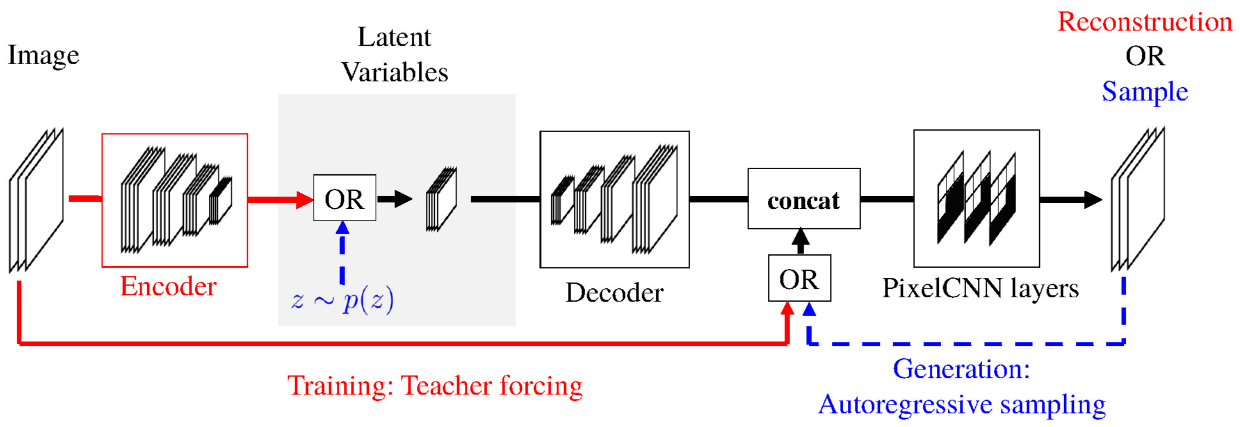


Figure 2.2: Our proposed model, PixelVAE, makes use of PixelCNN to model an autoregressive decoder for a VAE. VAEs, which assume (conditional) independence among pixels, are known to suffer from blurry samples, while PixelCNN, modelling the joint distribution, produces sharp samples, but lack a latent representation that might be more useful for downstream tasks. PixelVAE combines the best of both worlds, providing a meaningful latent representation, while producing sharp samples.

natural image datasets.

Another promising recent approach is *generative adversarial networks* (GANs) [23], which pit a generator network and a discriminator network against each other. Recent work has improved training stability [64, 73] and incorporated inference networks into the GAN framework [20, 19]. GANs generate more compelling samples compared to our work, but exhibit unstable training dynamics and are known to ignore modes of the data distribution [20]. Further, it is difficult to accurately estimate the data likelihood in GANs.

2.3 PixelVAE model

Like a VAE, our model jointly trains an “encode” inference network, which maps an image $\boldsymbol{x}$ to a posterior distribution over latent variables $\boldsymbol{z}$, and a “decode” generative network, which models a distribution over x conditioned on $\boldsymbol{z}$. The encoder and decoder networks are composed of a series of convolutional layers, respectively with strided convolutions for downsampling in the encoder and transposed convolutions for upsampling in the decoder.

As opposed to most VAE decoders that model each dimension of the output independently (for example, by modelling the output as a Gaussian with diagonal covariance), we use a conditional PixelCNN in the decoder. Our decoder models $\boldsymbol{x}$ as the product of each dimension $\boldsymbol{x}_i$ conditioned on all previous dimensions and the latent variable z:

$$p(\boldsymbol{x}|\boldsymbol{z}) = \prod_i p(\boldsymbol{x}_i|\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{i-1}, \boldsymbol{z})$$

We first transform $\boldsymbol{z}$ through a series of convolutional layers into feature maps with the same spatial resolution as the output image and then concatenate the resulting feature maps with the image. The resulting concatenated feature maps are then further processed by several PixelCNN masked convolutional layers and a final PixelCNN 256-way softmax output.

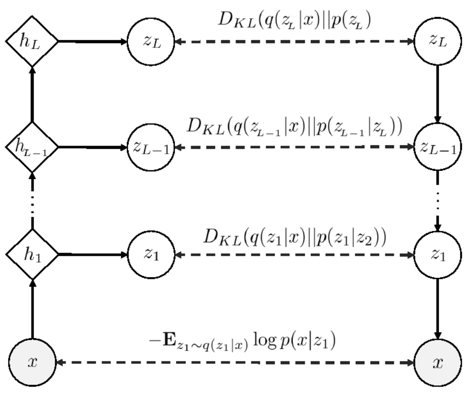

h_L
z_L
$D_{KL}(q(z_L|x)||p(z_L)$
z_L
h_{L-1}
z_{L-1}
$D_{KL}(q(z_{L-1}|x)||p(z_{L-1}|z_L))$
z_{L-1}
h_1
z_1
$D_{KL}(q(z_1|x)||p(z_1|z_2))$
z_1
$-\mathbf{E}_{z_1\sim q(z_1|x)} \log p(x|z_1)$
x
x

Figure 2.3: We generate top-down through a hierarchical latent space decomposition. Latent variables are inferred by composing successive deterministic functions to compute the parameters of the stochastic units. Dotted lines denote contributions to the cost.

Unlike typical PixelCNN implementations, we use very few PixelCNN layers in our decoder, relying on the latent variables to model the structure of the input at scales larger than the combined receptive field of our PixelCNN layers. As a result of this, our architecture captures global structure at a much lower computational cost than a standard PixelCNN implementation.

2.3.1 Hierarchical architecture

The performance of VAEs can be improved by stacking them to form a hierarchy of stochastic latent variables: in the simplest configuration, the VAE at each level models a distribution over the latent variables at the level below, with generation proceeding downward and inference upward through each level (as in Fig. 2.3). In convolutional architectures, the intermediate latent variables are typically organized into feature maps whose spatial resolution decreases toward higher levels.

Our model can be extended in the same way. At each level, the generator is a conditional PixelCNN over the latent features in the level below. This lets us autoregressively model not only the output distribution over pixels but also the prior over each set of latent feature maps. The higher-level PixelCNN decoders use diagonal Gaussian output layers instead of 256-way softmax, and model the dimensions within each spatial location (i.e. across feature maps) independently. This is done for simplicity, but is not a limitation of our model.

The output distributions over the latent variables for the generative and inference networks decompose as follows (see Fig. 2.3).

$$
\begin{aligned}
p(\boldsymbol{z}_1, \cdots, \boldsymbol{z}_L) &= p(\boldsymbol{z}_L)p(\boldsymbol{z}_{L-1}|\boldsymbol{z}_L)\cdots p(\boldsymbol{z}_1|\boldsymbol{z}_2)\\
q(\boldsymbol{z}_1, \cdots, \boldsymbol{z}_L|\boldsymbol{x}) &= q(\boldsymbol{z}_1|\boldsymbol{x})\cdots q(\boldsymbol{z}_L|\boldsymbol{x})
\end{aligned}
$$

We optimize the negative of the evidence lower bound (sum of data negative log-likelihood and KL-divergence of the posterior over latents with the prior).

$$
\begin{aligned}
-L(\boldsymbol{x}, q, p) &= -\underset{\boldsymbol{z}_1\sim q(\boldsymbol{z}_1|\boldsymbol{x})}{\mathbb{E}} \log p(\boldsymbol{x}|\boldsymbol{z}_1) + D_{KL}(q(\boldsymbol{z}_1, \cdots \boldsymbol{z}_L|\boldsymbol{x}) \;||\; p(\boldsymbol{z}_1, \cdots, \boldsymbol{z}_L))\\
&= -\underset{\boldsymbol{z}_1\sim q(\boldsymbol{z}_1|\boldsymbol{x})}{\mathbb{E}} \log p(\boldsymbol{x}|\boldsymbol{z}_1) + \int_{\boldsymbol{z}_1,\cdots,\boldsymbol{z}_L} \prod_{j=1}^{L} q(\boldsymbol{z}_j|\boldsymbol{x}) \sum_{i=1}^{L} \log \frac{q(\boldsymbol{z}_i|\boldsymbol{x})}{p(\boldsymbol{z}_i|\boldsymbol{z}_{i+1})} d\boldsymbol{z}_1 ... d\boldsymbol{z}_L
\end{aligned}
$$

$$
\begin{aligned}
&= -\underset{\boldsymbol{z}_1\sim q(\boldsymbol{z}_1|\boldsymbol{x})}{\mathbb{E}} \log p(\boldsymbol{x}|\boldsymbol{z}_1) + \sum_{i=1}^{L} \int_{\boldsymbol{z}_1,\cdots,\boldsymbol{z}_L} \prod_{j=1}^{L} q(\boldsymbol{z}_j|\boldsymbol{x}) \log \frac{q(\boldsymbol{z}_i|\boldsymbol{x})}{p(\boldsymbol{z}_i|\boldsymbol{z}_{i+1})} d\boldsymbol{z}_1 ... d\boldsymbol{z}_L \\
&= -\underset{\boldsymbol{z}_1\sim q(\boldsymbol{z}_1|\boldsymbol{x})}{\mathbb{E}} \log p(\boldsymbol{x}|\boldsymbol{z}_1) + \sum_{i=1}^{L} \int_{\boldsymbol{z}_i,\boldsymbol{z}_{i+1}} q(\boldsymbol{z}_{i+1}|\boldsymbol{x}) q(\boldsymbol{z}_i|\boldsymbol{x}) \log \frac{q(\boldsymbol{z}_i|\boldsymbol{x})}{p(\boldsymbol{z}_i|\boldsymbol{z}_{i+1})} d\boldsymbol{z}_i d\boldsymbol{z}_{i+1} \\
&= -\underset{\boldsymbol{z}_1\sim q(\boldsymbol{z}_1|\boldsymbol{x})}{\mathbb{E}} \log p(\boldsymbol{x}|\boldsymbol{z}_1) + \sum_{i=1}^{L} \underset{\boldsymbol{z}_{i+1}\sim q(\boldsymbol{z}_{i+1}|\boldsymbol{x})}{\mathbb{E}} \left[D_{KL}(q(\boldsymbol{z}_i|\boldsymbol{x}) \,||\, p(\boldsymbol{z}_i|\boldsymbol{z}_{i+1})) \right]
\end{aligned}
$$

Note that when specifying an autoregressive prior over each latent level $\boldsymbol{z}_i$, we can leverage masked convolutions [59] and samples drawn independently from the approximate posterior $q(\boldsymbol{z}_i|\boldsymbol{x})$ (i.e. from the inference network) to train efficiently in parallel on GPUs.

2.4 Experiments

2.4.1 MNIST

We evaluate our model on the binarized MNIST dataset [90, 42] and report results in Table 2.1. We also experiment with a variant of our model in which each PixelCNN layer is directly conditioned on a linear transformation of latent variable, $\boldsymbol{z}$ (rather than transforming $\boldsymbol{z}$ first through several upsampling convolutional layers) (as in Oord et al. [59] and find that this further improves performance, achieving an NLL upper bound comparable with the current state of the art. We estimate the marginal likelihood of our MNIST model using the importance sampling technique in [8], which computes a lower bound on the likelihood whose tightness increases with the number of importance samples per datapoint. We use $N = 5000$ samples per datapoint (higher values don't appear to significantly affect the likelihood estimate) and achieve state-of-the-art likelihood.

Model	**NLL Test**
DRAW [24]	≤ 80.97
Discrete VAE [70]	$= 81.01$
IAF VAE [36]	≈ 79.88
PixelCNN [57]	$= 81.30$
PixelRNN [57]	$= 79.20$
VLAE [12]	$= 79.03$
Convolutional VAE	≤ 87.41
PixelVAE	≤ 80.64
Gated PixelCNN (our implementation)	$= 80.10$
Gated PixelVAE	≈ 79.48 (≤ 80.02)
Gated PixelVAE without upsampling	$\approx$ **78.96** (≤ 79.58)

Table 2.1: We compare performance of different models on binarized MNIST. *PixelCNN* is the model described in Oord, Kalchbrenner, and Kavukcuoglu [57]. Our corresponding latent variable model is *PixelVAE*. *Gated PixelCNN* and *Gated PixelVAE* use the gated activation function in Oord et al. [59]. In *Gated PixelVAE without upsampling*, a linear transformation of latent variable conditions the (gated) activation in every PixelCNN layer instead of using upsampling layers.

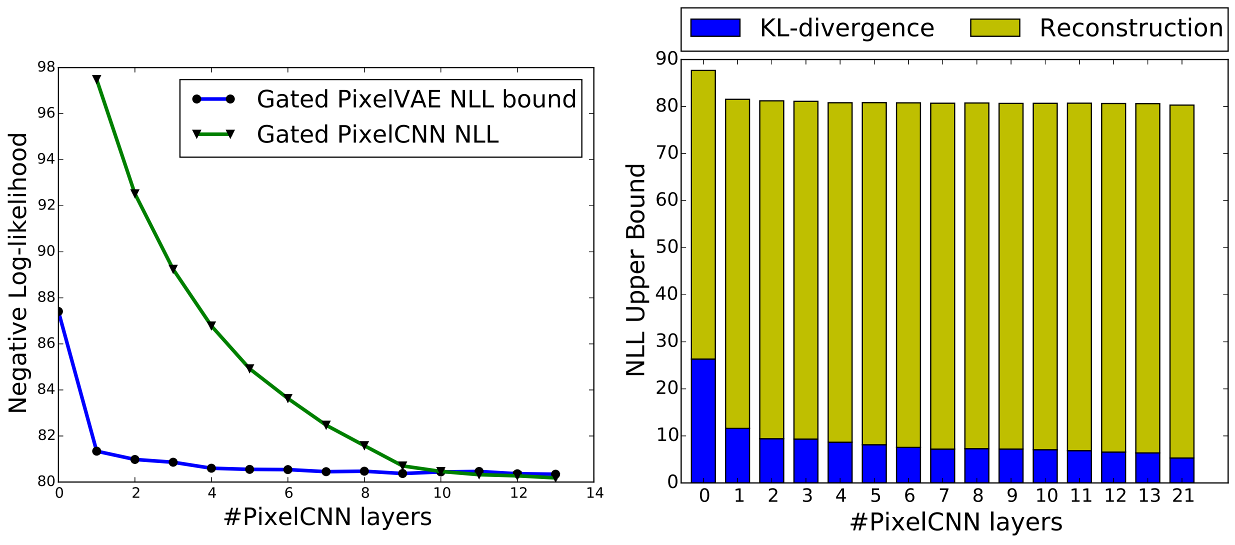


Figure 2.4: (a) Comparison of the negative loglikelihood upper bound of PixelVAE and exact NLL for PixelCNN as a function of the number of PixelCNN layers used. (b) Cost break down into KL-divergence and reconstruction cost.

Number of PixelCNN layers

The masked convolutional layers in PixelCNN are computationally expensive because they operate at the full resolution of the image and in order to cover the full receptive field of the image, PixelCNN typically needs a large number of them. One advantage of our architecture is that we can achieve strong performance with very few PixelCNN layers, which makes training and sampling from our model significantly faster than PixelCNN. To demonstrate this, we compare the performance of our model to PixelCNN as a function of the number of PixelCNN layers (Fig. 2.4a). We find that with fewer than 10 autoregressive layers, our PixelVAE model performs much better than PixelCNN. This is expected, since with few layers, the effective receptive field of the PixelCNN output units is not large enough to capture long-range dependencies in the data.

We also observe that adding the first PixelCNN layer has a dramatic impact on the NLL bound of PixelVAE. This is not surprising since the PixelCNN layer helps model local characteristics (with much greater expressiveness) which are complementary to the global characteristics which a VAE with a factorized output distribution models.

Latent variable information content

Because the autoregressive conditional likelihood function of PixelVAE is expressive enough to model some properties of the image distribution, it isn't forced to account for those properties through its latent variables as a standard VAE is. As a result, we can expect PixelVAE to learn latent representations which are invariant to textures, precise positions, and other attributes which are more efficiently modelled by the autoregressive decoder. To empirically validate this, we train PixelVAE models with different numbers of autoregressive layers (and hence, decoder receptive field sizes) and plot the breakdown of the NLL bound for each of these models into the reconstruction term $\log p(\boldsymbol{x}|\boldsymbol{z})$ and the KL divergence term $D_{KL}(q(\boldsymbol{z}|\boldsymbol{x}) \mid\mid p(\boldsymbol{z}))$ (Fig. 2.4b). The KL divergence term can be interpreted as a measure of the information content in the posterior distribution $q(\boldsymbol{z}|\boldsymbol{x})$ (in the sense that in expectation, samples from $q(\boldsymbol{z}|\boldsymbol{x})$ require $KL(q \mid\mid p)$ fewer bits to code under a

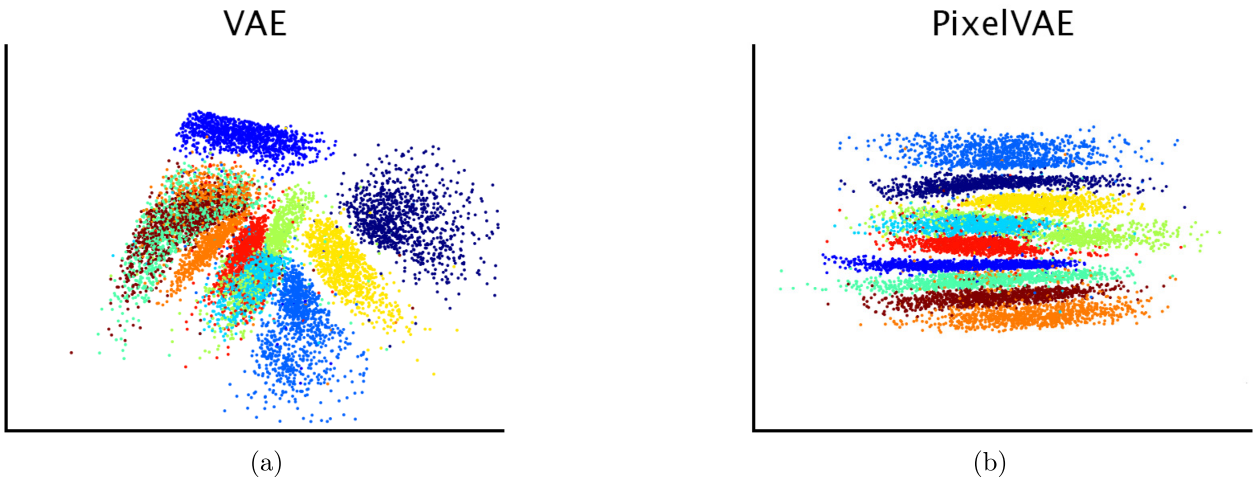


Figure 2.5: Visualization of the MNIST test set in the latent space of (a) convolutional VAE and (b) PixelVAE with two latent dimensions. PixelVAE separates classes more completely than VAE.

code optimized for q than under one optimized for p [9]) and hence, models with smaller KL terms encode less information in their latent variables.

We observe a sharp drop in the KL divergence term when we use a single autoregressive layer compared to no autoregressive layers, indicating that the latent variables have been freed from having to encode small-scale details in the images. Since the addition of a single PixelCNN layer allows the decoder to model interactions between pixels which are at most 2 pixels away from each other (our masked convolution filter size is 5×5), we can also say that most of the non-trivial, global structure in the images is still encoded in the latent variables.

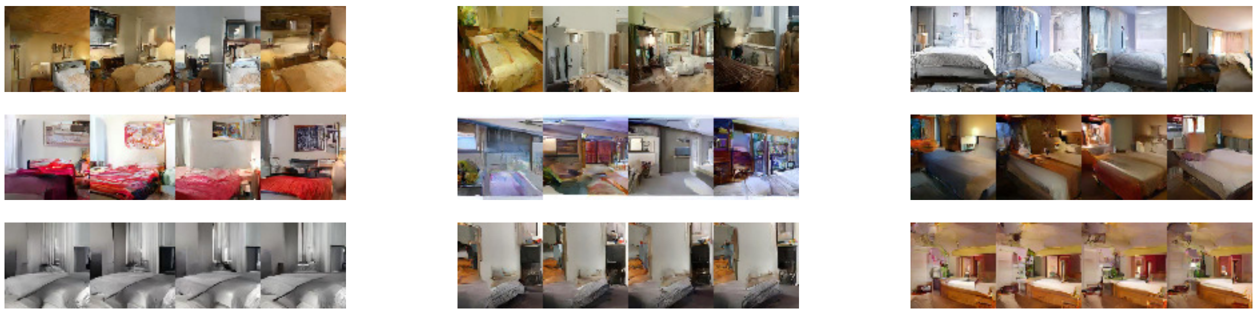


Figure 2.6: We visually inspect the variation in image features captured by the different levels of decoding in our model. For the two-level latent variable model trained on 64×64 LSUN bedrooms, we vary only the top-level sampling noise *(top)* while holding the other levels constant, vary only the middle-level noise *(middle)*, and vary only the bottom (pixel-level) noise *(bottom)*. From a visual inspection, we observe an alignment of model behaviour with model design: it seems that the top-level latent variables learn to model room structure and overall geometry, the middle-level latents model colour and texture features, and the pixel-level distribution models low-level image characteristics such as texture, colour, shading.

Latent representations

On MNIST, given a sufficiently high-dimensional latent space, VAEs have already been shown to learn representations in which digits are well-separated [77]. However, this task becomes more challenging as the capacity of the latent space is decreased. PixelVAE's flexible output distribution should allow it to learn a latent representation which is invariant to small details and thus better model global factors of variation given limited capacity.

To test this, we train a PixelVAE with a two-dimensional latent space, and an equivalent VAE. We visualize the distribution of test set images in latent space and observe that PixelVAE's latent representation separates digits significantly better than VAE (Figure 2.5). To quantify this difference, we train a K-nearest neighbors classifier in the latent space of each model and find that PixelVAE significantly outperforms VAE, achieving a test error of 7.2% compared to VAE's 22.9%. We also note that unlike VAE, PixelVAE learns a representation in which digit identity is largely disentangled from other latent causes.

2.4.2 LSUN bedrooms

To evaluate our model's performance with more data and complicated image distributions, we perform experiments on the LSUN bedrooms dataset [93]. We use the same preprocessing as in Radford, Metz, and Chintala [64] to remove duplicate images in the dataset. For quantitative experiments we use a 32×32 downsampled version of the dataset, and for qualitative demonstration, we present samples from a model trained on the 64×64 version.

We train a two-level PixelVAE with latent variables at 1×1 and 8×8 spatial resolutions. We find that this outperforms both a two-level convolutional VAE with diagonal Gaussian output and a single-level PixelVAE in terms of loglikelihood and sample quality. We also try replacing the PixelCNN layers at the higher level with a diagonal Gaussian decoder and find that this hurts loglikelihood, which suggests that multi-scale PixelVAE uses those layers effectively to autoregressively model latent features.

Features modeled at each layer

To see which features are modeled by each of the multiple layers, we draw multiple samples while varying the sampling noise at only a specific layer (either at the pixel-wise output or one of the latent layers) and visually inspect the resulting images (Fig. 2.6). When we vary only the pixel-level sampling (holding $\boldsymbol{z}_1$ and $\boldsymbol{z}_2$ fixed), samples are almost indistinguishable and differ only in precise positioning and shading details, suggesting that the model uses the pixel-level autoregressive distribution to model only these features. Samples where only the noise in the middle-level (8×8) latent variables is varied have different objects and colours, but appear to have similar basic room geometry and composition. Finally, samples with varied top-level latent variables have diverse room geometry.

2.4.3 64×64 ImageNet

64×64 ImageNet generative modelling was introduced in Oord, Kalchbrenner, and Kavukcuoglu [57] and involves density estimation of a difficult, highly varied image distribution. We trained a heirarchical PixelVAE model (with a similar architecture to the model in section 2.4.2) on 64×64 ImageNet and report validation set likelihood in Table 2.2. Our model achieves a likelihood competitive with Oord, Kalchbrenner, and Kavukcuoglu [57] and Oord et al. [59], despite being substantially less computationally complex. A visual inspection of ImageNet samples from our

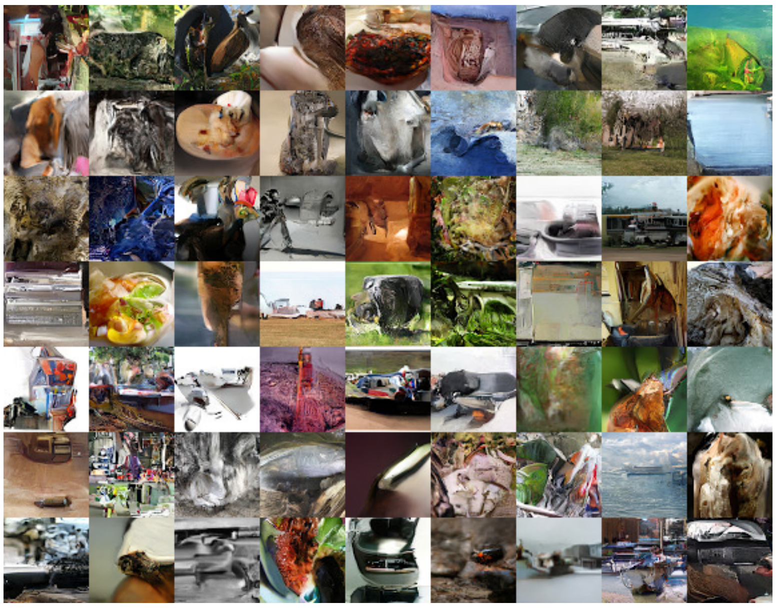


Figure 2.7: Samples from hierarchical PixelVAE on the 64x64 ImageNet dataset.

model (Fig. 2.7) also reveals them to be significantly more globally coherent than samples from PixelRNN.

Model	NLL Validation (Train)	FLOPs
Convolutional DRAW	≤ 4.10 (4.04)	—
Real NVP	$= 4.01$ (3.93)	—
PixelRNN	$= 3.63$ (3.57)	154×10^9
Gated PixelCNN	$=$ **3.57** (3.48)	134×10^9
Hierarchical PixelVAE	≤ 3.62 (3.55)	63×10^9

Table 2.2: Model performance on 64×64 ImageNet. We achieve competitive NLL at a fraction of the computational complexity of other leading models.

2.5 Conclusions

In this paper, we introduced a VAE model for natural images with an autoregressive decoder that achieves strong performance across a number of datasets. We explored properties of our model, showing that it can generate more compressed latent representations than a standard VAE and that it can use fewer autoregressive layers than PixelCNN. We established a new state-of-the-art on binarized MNIST in terms of loglikelihood, we achieved strongly competitive loglikelihood on 64×64 ImageNet, and demonstrated that our model generates high-quality samples on LSUN bedrooms.

The ability of PixelVAE to learn compressed representations in its latent variables by ignoring the small-scale structure in images is potentially very useful for downstream tasks. It would be interesting to further explore our model's capabilities for semi-supervised classification and representation learning in future work.

Acknowledgments

The authors would like to thank the developers of Theano [81] and Blocks and Fuel [50]. We acknowledge the support of the following agencies for research funding and computing support: Ubisoft, Nuance Foundation, NSERC, Calcul Quebec, Compute Canada, CIFAR, MEC Project TRA2014-57088-C2-1-R, SGR project 2014-SGR-1506 and TECNIOspring-FP7-ACCIÓ grant.

Prologue to the second article

Title: *Improved training of Wasserstein GANs.*
Authors: Ishaan Gulrajani, **Faruk Ahmed**, Martin Arjovsky, Vincent Dumoulin, Aaron Courville.

Abstract Generative adversarial networks *(GANs) are powerful generative models, but suffer from training instability. The recently proposed* Wasserstein GAN *(WGAN) makes progress toward stable training of GANs, but can still generate low-quality samples or fail to converge in some settings. We find that these problems are often due to the use of weight clipping in WGAN to enforce a Lipschitz constraint on the critic, which can lead to pathological behavior. We propose an alternative to clipping weights: penalize the norm of gradient of the critic with respect to its input. Our proposed method performs better than standard WGAN and enables stable training of a wide variety of GAN architectures with almost no hyperparameter tuning, including 101-layer ResNets and language models over discrete data. We also achieve high quality generations on CIFAR-10 and LSUN bedrooms. Code for our models is available at https://github.com/igul222/improved_wgan_training.*

A short version of this article was accepted at the ICML 2017 workshop on *Learning in Implicit Generative Models*, and the extended version has been accepted at the Neural Information Processing Systems conference (2017) as a Poster.

The idea was Ishaan Gulrajani's who also led the project and wrote the majority of the code-base. My personal contributions were:

- performing the experiments that study the limitations of weight clipping, specifically those that demonstrate the sensitivity of gradient norm magnitudes on the clipping parameter, and the saturation of weights with clipping,
- experimenting with hyper-parameter tuning and architectural variants,
- running experiments with CIFAR-10, and comparing convergence plots with baselines (WGAN-clipping and DC-GAN),
- initial experiments with loss curves in the context of overfitting in GANs,
- contributing to the writing and presentation of the paper.

Chapter 3

Improved training of Wasserstein GANs

3.1 Introduction

Generative adversarial networks (GANs) [23] are a powerful class of generative models that cast generative modelling as a game between two networks: a generator network produces synthetic data given some noise source and a discriminator network discriminates between the generator's output and true data. GANs can produce very visually appealing samples, but are often hard to train, and much of the recent work on the subject [73, 51, 3, 63] has been devoted to finding ways of stabilizing training. Despite this, consistently stable training of GANs remains an open problem.

In particular, [2] provides an analysis of the convergence properties of the value function being optimized by GANs. Their proposed alternative, named *Wasserstein GAN* (WGAN) [3], leverages the Wasserstein distance to produce a value function which has better theoretical properties than the original. WGAN requires that the discriminator (called the *critic* in that work) must lie within the space of 1-Lipschitz functions, which the authors enforce through weight clipping.

Our contributions are as follows:

1. On toy datasets, we show how critic weight clipping can lead to pathological behaviour,
2. We propose *WGAN with gradient penalty*, which does not suffer from the same issues,
3. We demonstrate stable training of many difficult GAN architectures with default settings, performance improvements over weight clipping, high-quality generations on CIFAR-10 and LSUN bedrooms, and a character-level GAN language model with a very simple architecture.

3.2 Background

3.2.1 Generative adversarial networks

The GAN training strategy is to define a game between two competing networks. The *generator* network maps a source of noise to the input space. The *discriminator* network receives either a generated sample or a true data sample and must distinguish between the two. The generator is trained to fool the discriminator.

Formally, the game between the generator G and the discriminator D is the minimax objective:

$$\min_G \max_D \mathop{\mathbb{E}}_{\boldsymbol{x}\sim\mathbb{P}_r}[\log(D(\boldsymbol{x}))] + \mathop{\mathbb{E}}_{\tilde{\boldsymbol{x}}\sim\mathbb{P}_g}[\log(1 - D(\tilde{\boldsymbol{x}}))], \tag{3.1}$$

where $\mathbb{P}_r$ is the data distribution and $\mathbb{P}_g$ is the model distribution implicitly defined by $\tilde{\boldsymbol{x}} = G(\boldsymbol{z})$, $\boldsymbol{z} \sim p(\boldsymbol{z})$ (the input $\boldsymbol{z}$ to the generator is sampled from some simple noise distribution, such as the uniform distribution or a spherical Gaussian distribution).

If the discriminator is trained to optimality before each generator parameter update, then minimizing the value function amounts to minimizing the Jensen-Shannon divergence between the data and model distributions on $\boldsymbol{x}$ [23], but doing so often leads to vanishing gradients as the discriminator saturates. In practice, [23] advocates that the generator be instead trained to maximize $\mathbb{E}_{\tilde{\boldsymbol{x}}\sim\mathbb{P}_g}[\log(D(\tilde{\boldsymbol{x}}))]$, which goes some way to circumvent this difficulty. However, even this modified loss function can misbehave in the presence of a good discriminator [2].

3.2.2 Wasserstein GANs

Arjovsky, Chintala, and Bottou [3] argue that the divergences which GANs typically minimize are potentially not continuous with respect to the generator's parameters, leading to training difficulty. They propose to instead use the *Earth-Mover* (also called Wasserstein-1) distance $W(q,p)$, which is informally defined as the minimum cost of transporting mass in order to transform the distribution q into the distribution p (where the cost is mass times transport distance). Under mild assumptions, $W(q,p)$ is continuous everywhere and differentiable almost everywhere.

The WGAN value function is constructed using the Kantorovich-Rubinstein duality [84] to obtain

$$\min_G \max_{D\in\mathcal{D}} \mathop{\mathbb{E}}_{\boldsymbol{x}\sim\mathbb{P}_r}\big[D(\boldsymbol{x})\big] - \mathop{\mathbb{E}}_{\tilde{\boldsymbol{x}}\sim\mathbb{P}_g}\big[D(\tilde{\boldsymbol{x}}))\big] \tag{3.2}$$

where $\mathcal{D}$ is the set of 1-Lipschitz functions and $\mathbb{P}_g$ is once again the model distribution implicitly defined by $\tilde{\boldsymbol{x}} = G(\boldsymbol{z}),\ \ \boldsymbol{z}\sim p(\boldsymbol{z})$. In that case, under an optimal discriminator (called a *critic* in the paper, since it's not trained to classify), minimizing the value function with respect to the generator parameters minimizes $W(\mathbb{P}_r,\mathbb{P}_g)$.

The WGAN value function results in a critic function whose gradient with respect to its input is better behaved than its GAN counterpart, making optimization of the generator easier. Additionally, WGAN has the desirable property that its value function correlates with sample quality, which is not the case for GANs.

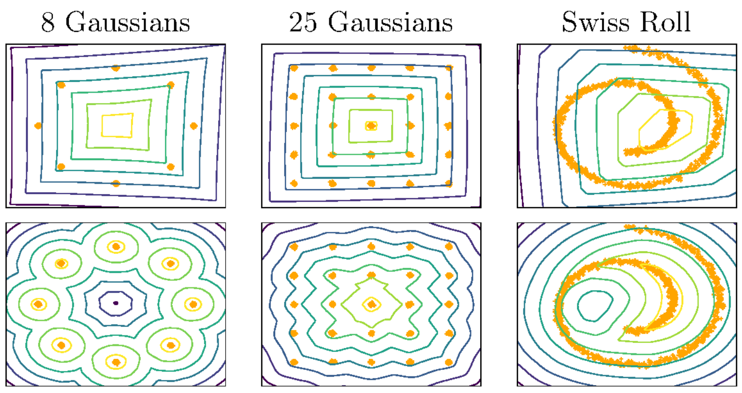
To enforce the Lipschitz constraint on the critic, [3] propose to clip the weights of the critic to lie within a compact space $[-c,\,c]$. The set of functions satisfying this constraint is a subset of the k-Lipschitz functions for some k which depends on c and the critic architecture. In the following sections, we demonstrate some of the issues with this approach and propose an alternative.

3.2.3 Properties of the optimal WGAN critic

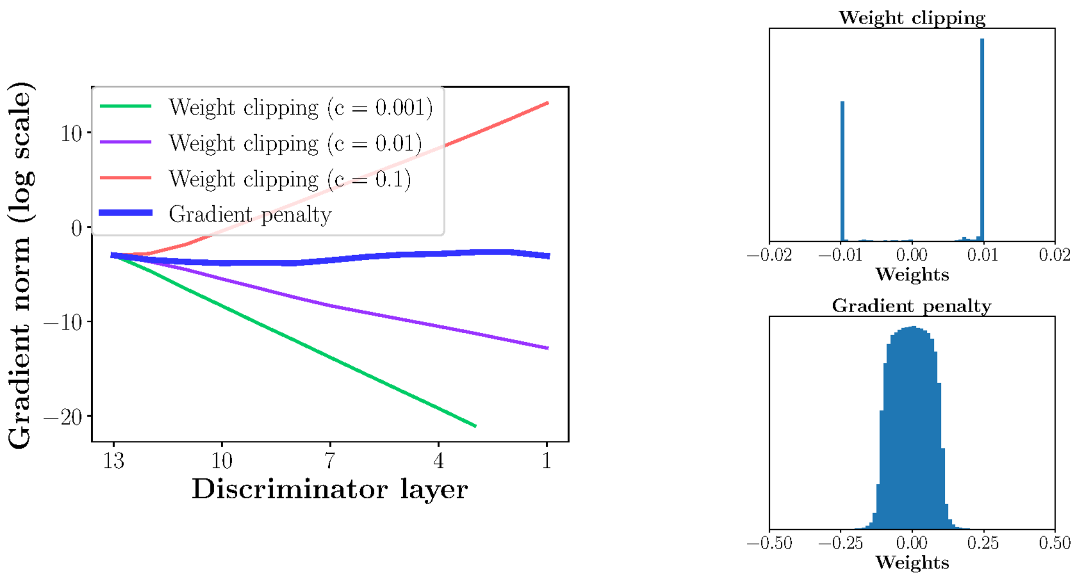
In order to understand why weight clipping is problematic in a WGAN critic, as well as to motivate our approach, we highlight some properties of the optimal critic in the WGAN framework. We more formally state and prove these in the Appendix.

If the optimal critic under the Kantorovich-Rubinstein dual D^* is differentiable, and x is a point from our generator distribution $\mathbb{P}_g$, then there is a point y sampled from the true distribution $\mathbb{P}_r$ such that the gradient of D^* at all points $x_t = (1-t)x + ty$ on a straight line between x and y points directly towards y, meaning $\nabla D^*(x_t) = \frac{y-x_t}{\|y-x_t\|}$.

This implies that the optimal critic has gradient norm 1 almost everywhere under $\mathbb{P}_r$ and $\mathbb{P}_g$.



(a) Value surfaces of WGAN critics trained to optimality on toy datasets using (top) weight clipping and (bottom) gradient penalty. Critics trained with weight clipping fail to capture higher moments of the data distribution. The 'generator' is held fixed at the real data plus Gaussian noise.



(b) (left) Gradient norms of deep WGAN critics during training on toy datasets either explode or vanish when using weight clipping, but not when using a gradient penalty. (right) Weight clipping (top) pushes weights towards two values (the extremes of the clipping range), unlike gradient penalty (bottom).

Figure 3.1: Gradient penalty in WGANs does not exhibit undesired behaviour like weight clipping.

3.3 Difficulties with weight constraints

We find that weight clipping in WGAN leads to optimization difficulties, and that even when optimization succeeds the resulting critic can have a pathological value surface. We explain these problems below and demonstrate their effects; however we do not claim that each one always occurs in practice, nor that they are the only such mechanisms.

Our experiments use the specific form of weight constraint from Arjovsky, Chintala, and Bottou [3] (hard clipping of the magnitude of each weight), but we also tried other weight constraints (L2 norm clipping, weight normalization), as well as soft constraints (L1 and L2 weight decay) and found that they exhibit similar problems.

To some extent these problems can be mitigated with batch normalization in the critic, which

[3] use in all of their experiments. However even with batch normalization, we observe that very deep WGAN critics often get stuck in a bad regime and fail to learn.

3.3.1 Capacity underuse

Implementing a k-Lipshitz constraint via weight clipping biases the critic towards much simpler functions. As stated previously, the optimal critic under the WGAN loss function has unit gradient norm almost everywhere; under a weight-clipping constraint, most neural network architectures can only attain their maximum gradient norm of k when they learn extremely simple functions.

To demonstrate this, we train WGAN critics with weight clipping to optimality on several toy distributions, holding the generator distribution $\mathbb{P}_g$ fixed at the real distribution plus unit-variance Gaussian noise. We plot value surfaces of the critics in Figure 3.1a. We omit batch normalization in the critic. In each case, the critic trained with weight clipping ignores higher moments of the data distribution and instead models very simple approximations to the optimal functions. In contrast, our approach does not suffer from this behaviour.

3.3.2 Exploding and vanishing gradients

We observe that the WGAN optimization process is difficult because of interactions between the weight constraint and the cost function, which inevitably result in either vanishing or exploding gradients, depending on the value of the clipping threshold c.

To demonstrate this, we train WGAN on the Swiss Roll toy dataset, varying the clipping threshold c in $[10^{-1}, 10^{-2}, 10^{-3}]$, and plot the norm of the gradient of the critic loss with respect to successive layers of activations. Both generator and critic are 12-layer ReLU MLPs without batch normalization. Figure 3.1b shows that for each of these values, the gradient either grows or decays exponentially as we move farther back in the network. We find our method results in more stable gradients that neither vanish nor explode, allowing training of more complicated networks.

3.4 Gradient penalty

We now propose an alternative way to enforce the Lipschitz constraint. A differentiable function is 1-Lipschtiz if and only if it has gradients with norm at most 1 everywhere, so we consider directly constraining the gradient norm of the critic's output with respect to its input.

To circumvent tractability issues, we enforce a soft version of the constraint with a penalty on the gradient norm for random samples $\hat{\boldsymbol{x}} \sim \mathbb{P}_{\hat{\boldsymbol{x}}}$. Our new objective is

$$L = \underbrace{\underset{\tilde{\boldsymbol{x}}\sim\mathbb{P}_g}{\mathbb{E}}\left[D(\tilde{\boldsymbol{x}})\right] - \underset{\boldsymbol{x}\sim\mathbb{P}_r}{\mathbb{E}}\left[D(\boldsymbol{x})\right]}_{\text{Original critic loss}} + \underbrace{\lambda \underset{\hat{\boldsymbol{x}}\sim\mathbb{P}_{\hat{\boldsymbol{x}}}}{\mathbb{E}}\left[(\|\nabla_{\hat{\boldsymbol{x}}} D(\hat{\boldsymbol{x}})\|_2 - 1)^2\right].}_{\text{Our gradient penalty}} \tag{3.3}$$

Sampling distribution We implicitly define $\mathbb{P}_{\hat{\boldsymbol{x}}}$ sampling uniformly along straight lines between pairs of points sampled from the data distribution $\mathbb{P}_r$ and the generator distribution $\mathbb{P}_g$. This is motivated by the fact that the graph of the optimal critic consists of straight lines connecting points from $\mathbb{P}_r$ and $\mathbb{P}_g$ (see subsection 3.2.3). Given that enforcing the unit gradient norm constraint everywhere is intractable, enforcing it only along these straight lines seems sufficient and experimentally results in good performance.

Penalty coefficient All experiments in this paper use $\lambda = 10$, which we found to work well

Algorithm 1 WGAN with gradient penalty. We use default values of $\lambda = 10$, $n_{\text{critic}} = 5$, $\alpha = 0.0001$, $\beta_1 = 0$, $\beta_2 = 0.9$.

```
Require: The gradient penalty coefficient $\lambda$, the number of critic iterations per generator iteration $n_{\text{critic}}$, the batch size $m$, Adam hyperparameters $\alpha, \beta_1, \beta_2$.
Require: initial critic parameters $w_0$, initial generator parameters $\boldsymbol{\theta}_0$.
 1: while $\boldsymbol{\theta}$ has not converged do
 2:     for $t = 1, ..., n_{\text{critic}}$ do
 3:         for $i = 1, ..., m$ do
 4:             Sample real data $\boldsymbol{x} \sim \mathbb{P}_r$, latent variable $\boldsymbol{z} \sim p(\boldsymbol{z})$, a random number $\epsilon \sim U[0, 1]$.
 5:             $\tilde{\boldsymbol{x}} \leftarrow G_{\boldsymbol{\theta}}(\boldsymbol{z})$
 6:             $\hat{\boldsymbol{x}} \leftarrow \epsilon \boldsymbol{x} + (1 - \epsilon)\tilde{\boldsymbol{x}}$
 7:             $L^{(i)} \leftarrow D_w(\tilde{\boldsymbol{x}}) - D_w(\boldsymbol{x}) + \lambda(\|\nabla_{\hat{\boldsymbol{x}}} D_w(\hat{\boldsymbol{x}})\|_2 - 1)^2$
 8:         end for
 9:         $w \leftarrow \text{Adam}(\nabla_w \frac{1}{m} \sum_{i=1}^{m} L^{(i)}, w, \alpha, \beta_1, \beta_2)$
10:     end for
11:     Sample a batch of latent variables $\{\boldsymbol{z}^{(i)}\}_{i=1}^{m} \sim p(\boldsymbol{z})$.
12:     $\boldsymbol{\theta} \leftarrow \text{Adam}(\nabla_{\boldsymbol{\theta}} \frac{1}{m} \sum_{i=1}^{m} -D_w(G_{\boldsymbol{\theta}}(\boldsymbol{z})), \boldsymbol{\theta}, \alpha, \beta_1, \beta_2)$
13: end while
```

across a variety of architectures and datasets ranging from toy tasks to large ImageNet CNNs.

No critic batch normalization Most prior GAN implementations [64, 73, 3] use batch normalization in both the generator and the discriminator to help stabilize training, but batch normalization changes the form of the discriminator's problem from mapping a single input to a single output to mapping from an entire batch of inputs to a batch of outputs [73]. Our penalized training objective is no longer valid in this setting, since we penalize the norm of the critic's gradient with respect to each input independently, and not the entire batch. To resolve this, we simply omit batch normalization in the critic in our models, finding that they perform well without it.

Our method works with normalization schemes which don't introduce correlations between examples. In particular, we recommend layer normalization [5] as a replacement for batch normalization.

Two-sided penalty We encourage the norm of the gradient to go towards 1 (two-sided penalty) instead of just staying below 1 (one-sided penalty). In practice, we found this to converge slightly faster and to better optima. Empirically, this seems not to constrain the critic too much, likely because the optimal WGAN critic has gradients with norm 1 almost everywhere under $\mathbb{P}_r$ and $\mathbb{P}_g$ and in portions of the region in between (see subsection 3.2.3).

3.5 Experiments

3.5.1 Architecture robustness

To demonstrate the stability of our method's training process, we train a wide variety of GAN architectures on the LSUN bedrooms dataset [93]. In addition to the baseline DCGAN architecture from [64], we choose six architectures which we think are difficult to train:

1. no BN and a constant number of filters in the generator, as in [3],
2. 4-layer 512-dim ReLU MLP generator, as in [3],
3. no normalization in either the discriminator or generator

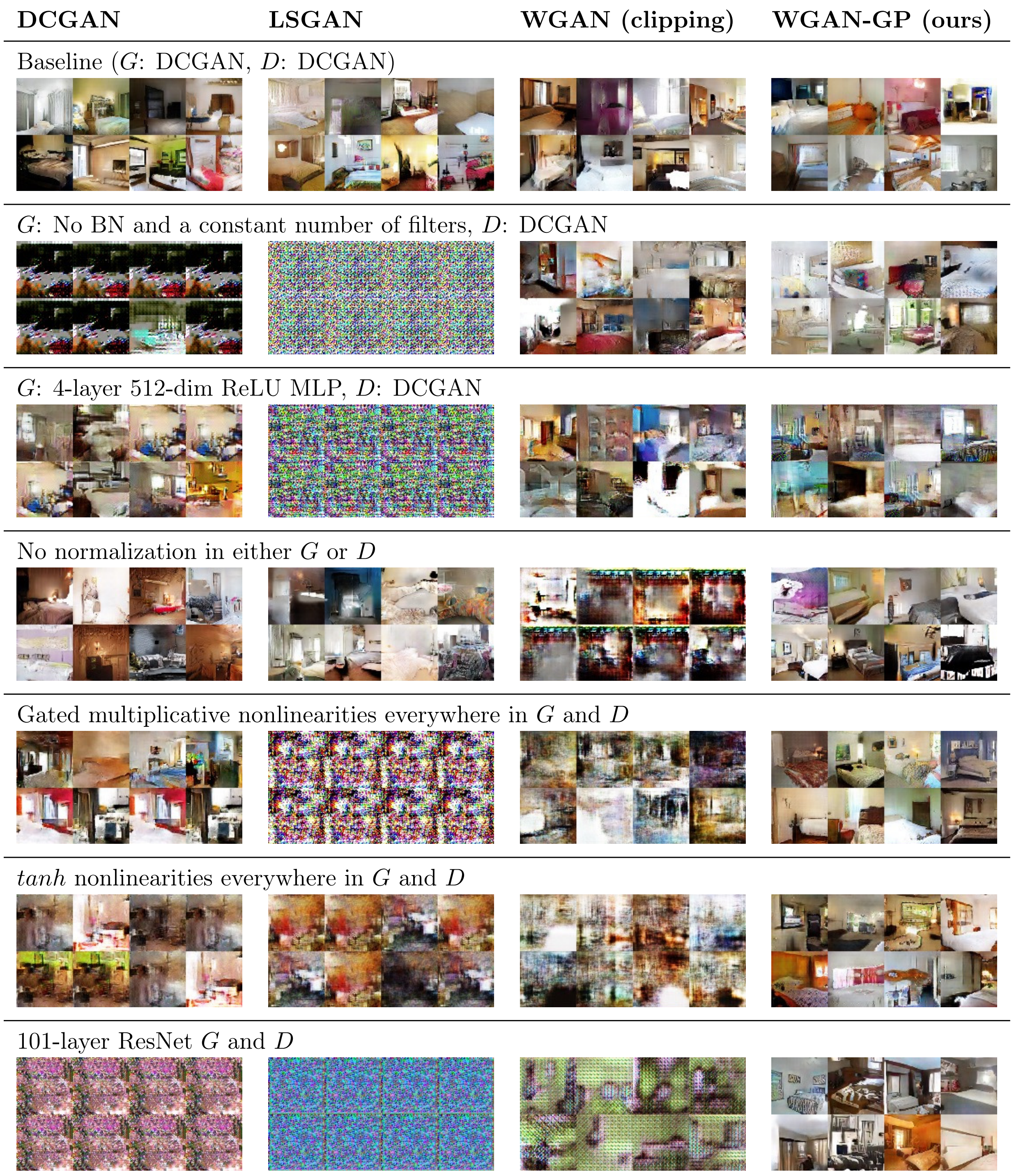

DCGAN
LSGAN
WGAN (clipping)
WGAN-GP (ours)
Baseline (G: DCGAN, D: DCGAN)
G: No BN and a constant number of filters, D: DCGAN
G: 4-layer 512-dim ReLU MLP, D: DCGAN
No normalization in either G or D
Gated multiplicative nonlinearities everywhere in G and D
tanh nonlinearities everywhere in G and D
101-layer ResNet G and D

Figure 3.2: Difficult GAN architectures trained with different methods. Only WGAN with gradient penalty succeeds in training every architecture with a single choice of default settings.

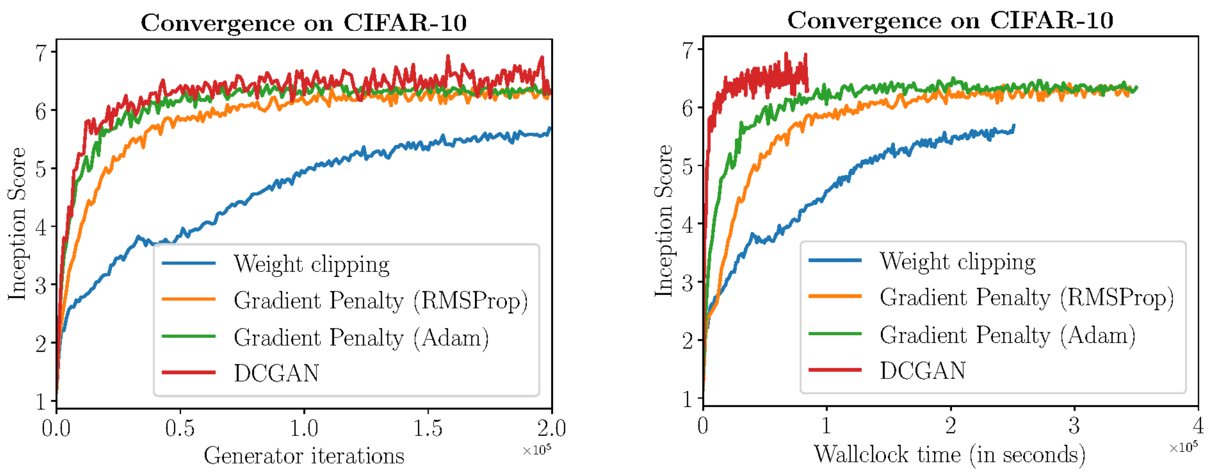


Figure 3.3: CIFAR-10 Inception score over generator iterations (left) or wall-clock time (right) for four models: WGAN with weight clipping, WGAN with gradient penalty and RMSProp (to control for the optimizer), WGAN with gradient penalty and Adam, and DCGAN. Even with the same learning rate, gradient penalty significantly outperforms weight clipping. DCGAN converges faster, but WGAN with gradient penalty achieves similar scores with improved stability.

4. gated multiplicative nonlinearities, as in [59],
5. *tanh* nonlinearities, and
6. 101-layer ResNet generator and discriminator.

Although we do not claim it is impossible without our method, to the best of our knowledge this is the first time very deep residual networks were successfully trained in a GAN setting. For each architecture, we train models using four different GAN procedures: WGAN with gradient penalty, WGAN with weight clipping, DCGAN [64], and Least-Squares GAN [49]. For each objective, we used the default set of optimizer hyperparameters recommended in that work (except LSGAN, where we searched over learning rates).

For WGAN with gradient penalty, we replace any batch normalization in the discriminator with layer normalization (see section 3.4). We train each model for 200K iterations and present samples in Figure 3.2. WGAN with gradient penalty is the only method which successfully trains every architecture with a single set of default hyperparameters. For every other training method, some of these architectures are unstable or suffer from mode collapse.

3.5.2 Improved performance over weight clipping

One advantage of our method over weight clipping is improved training speed and sample quality. To demonstrate this, we train WGANs with weight clipping and our gradient penalty on CIFAR-10 [40] and plot Inception scores [73] over the course of training (see Figure 3.3). For WGAN with gradient penalty, we train one model with the same optimizer (RMSProp) and learning rate as WGAN with weight clipping, and another model with Adam and a higher learning rate. Even with the same optimizer, our method converges faster and to a better final score than using weight clipping. Using Adam further improves performance. We also plot the performance of DCGAN [64] and observe that although our method converges more slowly (in terms of wall-clock time) than DCGAN, its performance is more stable at convergence.

Method	Score
ALI [20] (in [86])	$5.34 \pm .05$
BEGAN [6]	5.62
DCGAN [64] (in [31])	$6.16 \pm .07$
Improved GAN (-L+HA) [73]	$6.86 \pm .06$
EGAN-Ent-VI [16]	$7.07 \pm .10$
DFM [86]	$7.72 \pm .13$
WGAN-GP ResNet (ours)	$7.86 \pm .07$

Unsupervised Inception scores

Method	Score
SteinGAN [85]	6.35
DCGAN (with labels, in [85])	6.58
Improved GAN [73]	$8.09 \pm .07$
AC-GAN [56]	$8.25 \pm .07$
SGAN-no-joint [31]	$8.37 \pm .08$
WGAN-GP ResNet (ours)	$8.42 \pm .10$
SGAN [31]	$8.59 \pm .12$

Supervised Inception scores

Figure 3.4: Inception scores on CIFAR-10. Among unsupervised models, our model achieves state-of-the-art performance. With the addition of label information (following [56]) and no other tuning, our model outperforms all other supervised architectures except SGAN.

3.5.3 Sample quality on CIFAR-10 and LSUN bedrooms

For equivalent architectures, our method achieves comparable sample quality to the standard GAN objective. However the increased stability allows us to improve sample quality by exploring a wider range of architectures. To demonstrate this, we find an architecture which establishes a new state of the art Inception score on unsupervised CIFAR-10 (Figure 3.4). When we add label information (using the procedure in [56]), the same architecture outperforms all other published models except for SGAN.

We also train a deep ResNet on 128×128 LSUN bedrooms and show samples in Figure 3.6. We believe these samples are at least competitive with the best reported so far on any resolution for this dataset.

3.5.4 Character-level language modelling

Using GANs to model language is challenging, largely because text is a sequence of discrete tokens and discrete output units in the generator are difficult to backpropagate through. Most past attempts have addressed this with the REINFORCE gradient estimator [89] or continuous approximations to discrete sampling [34, 48], but have seen limited success without resorting to pretraining or joint training with a typical supervised maximum-likelihood objective [94, 46, 91, 10, 47]. [29] also propose a discrete GAN training method but do not evaluate on language modelling.

In contrast, using our method we train a GAN which generates discrete outputs without resorting to complicated "backprop through discrete variable" methods, maximum-likelihood training, or fine-tuned architectures. We train a character-level GAN language model on the Google Billion Word dataset [11]. Our generator is a simple CNN which deterministically transforms a latent vector into a sequence of 32 one-hot character vectors through 1D convolutions. We apply a softmax nonlinearity at the output, but use no sampling step: during training, the softmax output is passed directly into the critic (which likewise, is a simple 1D CNN). When decoding samples, we just take the argmax of each output vector.

We present samples from the model in Table 3.1. To our knowledge this is the first reported result of a general language model trained entirely adversarially without a supervised maximum-likelihood loss. Our model makes frequent spelling errors (likely because it has to output each character independently) but nonetheless manages to learn quite a lot about the statistics of language. Although

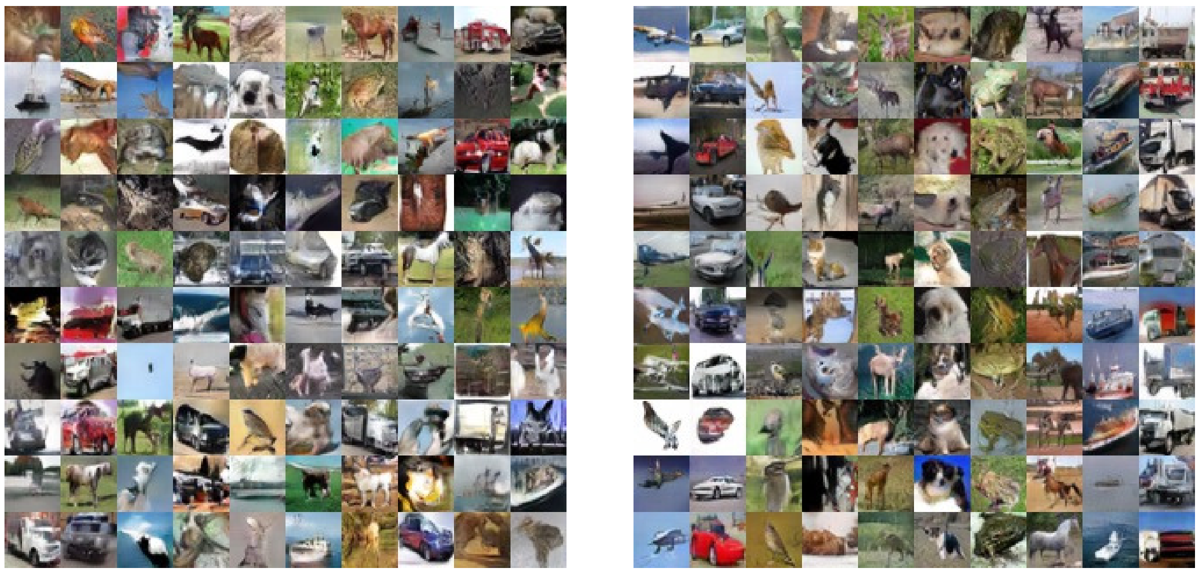


Figure 3.5: *(left)* CIFAR-10 samples generated by our unsupervised model. *(right)* Conditional CIFAR-10 samples, from adding AC-GAN conditioning to our unconditional model. Samples from the same class are displayed in the same column.

WGAN with gradient penalty (1D CNN)	
`Busino game camperate spent odea`	`Solice Norkedåin pring in since`
`In the bankaway of smarling the`	`ThiS record ( 31. ) UBS ) and Ch`
`SingersMay , who kill that imvic`	`It was not the annuas were plogr`
`Keray Pents of the same Reagun D`	`This will be us , the ect of DAN`
`Manging include a tudancs shat "`	`These leaded as most-worsd p2 a0`
`His Zuith Dudget , the Denàmbern`	`The time I paid0a South Cubry i`
`In during the Uitational questio`	`Dour Fraps higs it was these del`
`Divos from The ’ noth ronkies of`	`This year out howneed allowed lo`
`She like Monday , of macunsuer S`	`Kaulna Seto consficutes to repor`

Table 3.1: Samples from a WGAN character-level language model trained with our method on sentences from the Billion Word dataset, truncated to 32 characters. The model learns to directly output one-hot character embeddings from a latent vector without any discrete sampling step. In our experiments, the standard GAN objective with the same architecture collapsed to give very poor samples. With additional heuristics (such as pretraining) or very careful finetuning the standard GAN objective is known to avoid complete failure, but is acknowledged to fail out-of-the-box on discrete output spaces.

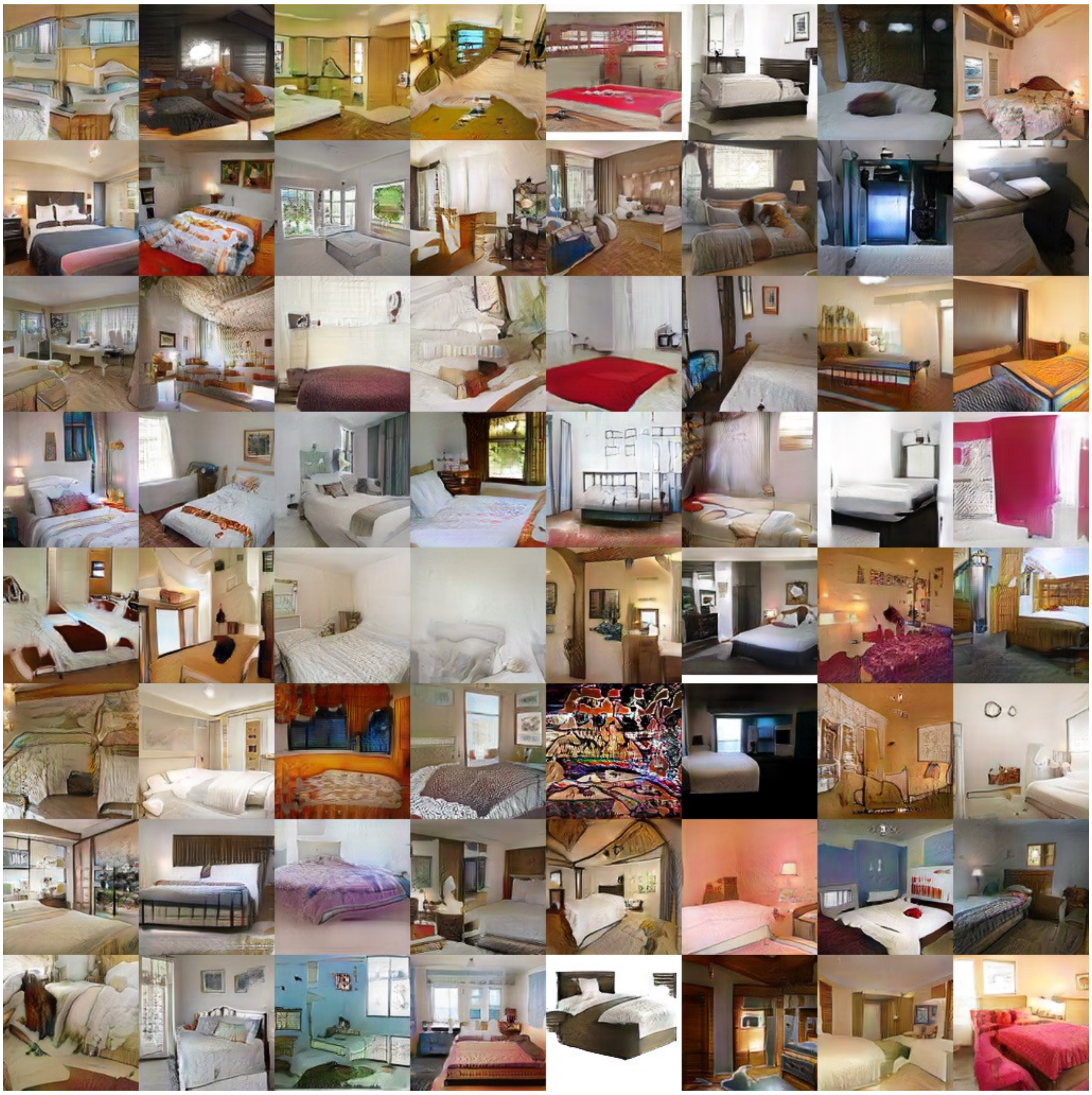
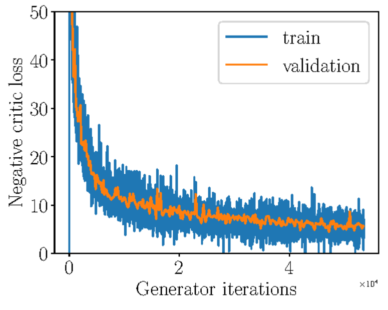
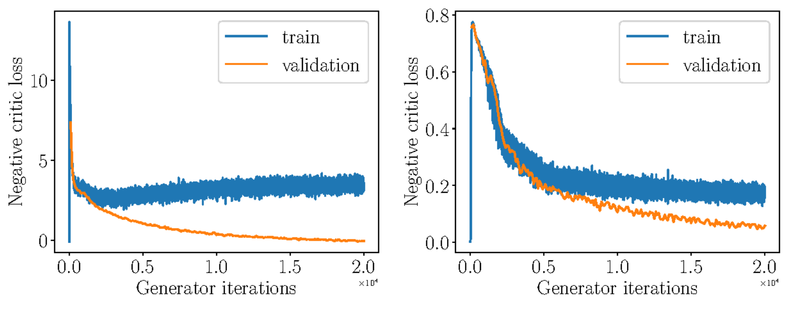


Figure 3.6: Samples of 128 × 128 LSUN bedrooms. We believe these samples are at least comparable to the best published results so far.



(a)



(b)

Figure 3.7: (a) The negative critic loss of our model on LSUN bedrooms converges toward a minimum as the network trains. (b) WGAN training and validation losses on a random 1000-digit subset of MNIST show overfitting when using either our method (left) or weight clipping (right). In particular, with our method, the critic overfits faster than the generator, causing the training loss to increase gradually over time even as the validation loss drops.

we do not claim that doing so is impossible, we were unable to produce comparable results with the standard GAN objective and a simple architecture.

The difference in performance between WGAN and other GANs can be explained by a simple fact. Consider the simplex $\Delta_n = \{p \in \mathbb{R}^n : p_i \geq 0, \sum_i p_i = 1\}$, and the set of vertices on the simplex (or one-hot vectors) $V_n = \{p \in \mathbb{R}^n : p_i \in \{0,1\}, \sum_i p_i = 1\} \subseteq \Delta_n$. If we have a vocabulary of size n and we have a distribution $\mathbb{P}_r$ over sequences of size T, we have that $\mathbb{P}_r$ is a distribution on $V_n^T = V_n \times \cdots \times V_n$. Since V_n^T is a subset of Δ_n^T, we can also treat $\mathbb{P}_r$ as a distribution on Δ_n^T (by assigning zero probability mass to all points not in V_n^T).

Now the interesting part is this: $\mathbb{P}_r$ is discrete (or supported on a finite number of elements, namely V_n^T) on Δ_n^T, but $\mathbb{P}_g$ can easily be a continuous distribution over Δ_n^T. The KL divergences between two such distributions are infinite, and so the JS divergence is saturated. In practice, this means a discriminator will quickly learn to reject all samples that don't lie on V_n^T (sequences of one-hot vectors), and the gradients passed to the generator are meaningless. However, it is easily seen that the conditions of Theorem 1 and Corollary 1 of [3] are satisfied even on this non-standard learning scenario with $\mathcal{X} = \Delta_n^T$. This means that $W(\mathbb{P}_r, \mathbb{P}_g)$ is still well defined, continuous everywhere and differentiable almost everywhere, and we can optimize it just like in any other continuous variable setting. The way this manifests is that in WGANs, the Lipschitz constraint forces the critic to provide a linear gradient from all Δ_n^T towards towards the real points in V_n^T.

3.5.5 Meaningful loss curves and detecting overfitting

An important benefit of weight-clipped WGANs is that their loss correlates with sample quality and converges towards a minimum. To show that our method preserves this property, we train a WGAN with gradient penalty on the LSUN bedrooms dataset [93] and plot the negative of the critic's loss in Figure 3.7a. We see that the loss converges as the generator minimizes $W(\mathbb{P}_r, \mathbb{P}_g)$.

GANs, like all models trained on limited data, will eventually overfit, but the lack of a meaningful loss metric has made overfitting hard to detect in the past. To explore the loss curve's behaviour when the network overfits, we train large unregularized WGANs on a random 1000-image subset of MNIST and plot the negative critic loss on both the training and validation sets in Figure 3.7b.

In both WGAN and WGAN-GP, the two losses diverge, suggesting that the critic overfits and provides an inaccurate estimate of $W(\mathbb{P}_r, \mathbb{P}_g)$, at which point all bets are off regarding correlation with sample quality. However in WGAN with gradient penalty, the training loss gradually increases even while the validation loss drops.

Wu et al. [90] also measure overfitting in GANs by estimating the generator's log-likelihood. Compared to that work, our method detects overfitting in the critic (rather than the generator), is faster and easier to implement, and measures the same loss that the network minimizes.

3.6 Conclusion

In this work, we demonstrated problems with weight clipping in WGAN and introduced an alternative in the form of a penalty term in the critic loss which does not exhibit the same problems. Using our method, we demonstrated strong modelling performance and stability across a variety of architectures.

Now that we have a more stable algorithm for training GANs, we hope our work opens the path for stronger modelling performance on large-scale image datasets and language. Another interesting direction is adapting our penalty term to the standard GAN objective function, where it might stabilize training by encouraging the discriminator to learn smoother decision boundaries.

Acknowledgements

We would like to thank Mohamed Ishmael Belghazi, Léon Bottou, Zihang Dai, Stefan Doerr, Ian Goodfellow, Kyle Kastner, Kundan Kumar, Luke Metz, Alec Radford, Sai Rajeshwar, Aditya Ramesh, Tom Sercu, Zain Shah and Jake Zhao for insightful comments. We would also like to thank the developers of Blocks and Fuel [50]. We acknowledge the support of the following agencies for research funding and computing support: Ubisoft, Nuance Foundation, NSERC, Calcul Quebec, Compute Canada, CIFAR, MEC Project TRA2014-57088-C2-1-R, SGR project 2014-SGR-1506 and TECNIOspring-FP7-ACCIÓ grant.

Prologue to the third article

Title: *Evaluating generative models as models of normality.*
Authors: Faruk Ahmed, Aaron Courville.

Abstract *We explore the potential of generative models as models of normality for images. We evaluate them in the context of* zero-shot out-of-distribution detection*, motivated by the potential problem of distributional change faced by AI systems deployed in the real world. We evaluate performance of VAE, PixelCNN, PixelVAE in a setup where the model is trained on all but a subset of data (corresponding to a class label) for MNIST and CIFAR-10, and compare likelihoods for in-distribution and out-of-distribution test data. We also evaluate DCGAN, by using the discriminator prediction for real/fake data as a surrogate measure of normality.*

This is work in progress, and has not been submitted to any venue as of the time of writing.

The initial idea for the held-out training setup was Aaron Courville's. I explored the idea further, established the experimental setup with context, performed all experiments, and wrote the following chapter.

Chapter 4

Evaluating generative models as models of normality

4.1 Introduction

In this chapter, we are motivated by the issue of *distributional change* [1]. A system which has been trained on data sampled from a specific distribution, might encounter situations in the real world where the data comes from a distribution that is different from the training distribution. In such circumstances it is very likely that the system will take a misguided action. Additionally, AI classification systems are known to provide incorrect predictions with high confidence [79, 1], which could potentially be unsafe for humans when these systems are operating in our midst. As a contrived example, a self-driving car that has never observed a wheelchair in the training data might confuse one for a vehicle and take potentially dangerous decisions about how to adapt its velocity. For a lot of such practical AI systems, it might not be possible to train them on highly inclusive data with detailed annotations (or extraneous information not directly pertaining to the task at hand). The nature of error could also be something more unexpected, other than a simple class-confusion as in the example. With this in mind, we shall operate under the assumption that all we have access to is the raw data, and no meta-information such as object annotations.

Our setting is *zero-shot out-of-distribution detection*: the system is given access to examples from a particular distribution of interest during training. At test time, it will encounter examples that are unfamiliar given what we have exposed the model to. The system is required to identify these novel instances as such, with reference to the learned distribution of the data presented during training. Additionally, we shall not assume access to any meta-information about the examples, such as labels of objects present in them. This is a difficult problem because the model is not told in advance what to look for – only to learn features for given data with a surrogate goal in mind, for example optimizing for good reconstruction from the feature representation in an autoencoder setting, or maximizing the marginal likelihood in a latent variable model. Without specific pressure which would emphasize particularly discriminative features of the given data, the model can well learn very general characteristics that are mostly shared with certain out-of-distribution examples that might be encountered in the wild.

Normality model [1] and detection module A *normality model* is a model that captures the essence of what constitutes "normal" data and can be used to identify samples that are deemed abnormal, or out-of-distribution. One approach to this task, borrowed from anomaly detection techniques, can be to learn a normality model, and couple this with a *detection module*. The detection module can query the normality model for degrees of normality of an input data point, and use

[1]The term is borrowed from the anomaly detection literature, particularly in work on security systems, and is not to be confused with *normality testing*, which tests if data fits a Gaussian distribution well.

the model response to classify the input as being in-distribution or out-of-distribution. The normality models in this framework are typically density estimates, or unsupervised clustering models that form clusters of feature representations and report distances to these clusters, or other alternative unary classification systems [80]. The detection module performs the final step of predicting a normal/abnormal label for a test example, by thresholding estimates of normality for example, or using some other transformations on the output of the normality module.

Objective We have seen in this thesis that modern generative models based on deep neural networks perform very well for generating images (with the highest likelihood scores and sample quality), so it seems reasonable to expect that they would be useful for out-of-distribution detection if appointed as normality models. In this chapter, we explore this possibility.

4.2 Experimental setup

We shall train candidate generative models on two simple image datasets, and then inspect if the models assign higher numbers denoting "belonging" for samples from the training distribution than for samples from a different distribution. For a generative model that has learned a good approximation to the training distribution, out-of-distribution samples should ideally be assigned lower scores.

4.2.1 Datasets

We shall report results of experiments on two datasets – MNIST [42] and CIFAR-10 [40]. For both datasets, we shall do the following for testing a generative model: we shall train ten versions of the model, each time leaving out images corresponding to one of the image categories (digits in the case of MNIST, object classes in the case of CIFAR-10); then we shall compute the most straightforward normality measure as reported by the trained generative model (for example, in the case of explicit models, this is log-likelihood) on two sets of the test set images, one with the nine classes that were trained on, and the other with the held-out class. We shall then quantify the relative ranking of normality scores for the out-of-distribution vs. the in-distribution test data.

4.2.2 Models

The explicit models (refer to subsection 1.3.2) we shall evaluate are VAE, PixelCNN, and PixelVAE. It is easy to implement these as normality models since they provide us with log-likelihoods (or lower bound estimates to log-likelihoods). We shall also evaluate a GAN model, specifically the popular DCGAN, because although it is an implicit model with a highly non-trivial log-likelihood estimate, we can use the discriminator's decision as a surrogate. However, this is certainly not an obvious choice: since the discriminator is trained to differentiate real from generated data, little can be assumed about the precise nature of this decision boundary in the context of generalizing to an in-distribution vs. out-of-distribution classification task.

MNIST architectures We binarize the digits, since most modern generative models evaluate on the binarized version, which is easier to train using cross-entropy. We do not use the standard binarized version from Salakhutdinov and Murray [72] because labels are not provided for that release, but dynamically binarize the digits of the original version at run-time. For the VAE implementation, we use a convolutional encoder and decoder with 8 layers each and 3×3 filters (following Simonyan and Zisserman [76]). The PixelCNN and PixelVAE architectures are the ones discussed in chapter 2. For the DCGAN, we do not binarize the digits, and follow the architecture in [64], except we also implement batch-normalization in the generator, following current practice, but omit it in the discriminator in order to prevent batch statistics from providing an indirect signal about the distribution. Even if we use population statistics, it's not clear which distribution to use as the population – the real data or the generated data, since training was done with batch statistics from both distributions.

CIFAR-10 architectures A plain convolutional VAE implementation is known to perform very poorly for CIFAR images, so we make use of a specific variant called *inverse autoregressive flow* (IAF) which produces less blurry samples.

Rezende and Mohamed [67] introduced the idea of *normalizing flows*, which transforms a draw from a simple distribution into a draw from a complex one by successive invertible transformations. Kingma et al. [38] implement a particular normalizing flow as a chain of inverses of autoregressive Gaussian invertible transforms [2] which is used to express the approximate posterior in a VAE. This equips the VAE with a more flexible posterior than a diagonal Gaussian, and we use this IAF-VAE in our experiments with CIFAR-10.

A PixelCNN implementation did not work well for CIFAR-10 in our experiments (in terms of inspected sample quality), so we used an improved version for the CIFAR-10 experiments: *PixelCNN++* [74], in which a few improvements are made to PixelCNN, such as using a logistic mixture model on the pixel space, predicting all channels of a pixel at once (with a simpler factorization over the channels of the center pixel), and a few other architectural changes.

PixelVAE and DCGAN produce reasonable samples for the standard architectures.

4.2.3 Evaluation

For a binary classification task, when the two classes are imbalanced at evaluation time, accuracy is not a very good metric for a classifier's performance, since accuracy can be misleadingly high by simply labeling everything the majority class. In such cases, using performance measures that take into account how often the classifier is correct when it assigns a positive label, and how often it correctly identifies the positive class [3] are more useful and informative. These two quantites are expressed as *precision* and *recall*. Specifically,

$$\text{precision} = \frac{\text{true positives}}{\text{true positives} + \text{false positives}}, \tag{4.1}$$

$$\text{recall} = \frac{\text{true positives}}{\text{true positives} + \text{false negatives}}. \tag{4.2}$$

A precision-recall (PR) curve is defined as a set of precision-recall points

$$\text{PR curve} \triangleq \{\text{recall}(t), \text{precision}(t), -\infty < t < \infty\}, \tag{4.3}$$

where t is a threshold parameter. We shall consider our normality module as producing output values such that larger output values are associated with positive examples (so we would need to negate log-likelihood numbers). For a given threshold t, if the output of the normality module is greater than t, it shall be labeled positive.

[2]Using the inverses of the invertible transforms instead of the forward path comes with computational benefits: now the dimensions of the complex posterior can be computed in parallel, and the mapping is equally powerful in terms of the amount of "complexity" it induces upon the simple base distribution.

[3]We shall assume that the less frequent class (in our case, the out-of-distribution population) is labeled positive, and is of interest; and that the two distributions corresponding to the positve and negative class are conditionally independent given the class.

OOD digit	Skew	VAE	PixCNN	PixelVAE	DC-GAN
0	9.8	40.20	31.31	40.89	44.16
1	11.35	5.98	5.98	6.08	6.00
2	10.32	65.37	42.60	56.12	14.40
3	10.1	36.86	36.31	38.33	12.92
4	9.82	34.49	19.74	19.09	7.38
5	8.92	38.56	26.29	28.12	9.75
6	9.58	38.13	23.76	33.16	9.68
7	10.28	10.13	8.33	11.75	7.17
8	9.74	57.64	50.57	54.85	12.87
9	10.09	11.09	12.49	11.50	8.23
mean	10	**33.85**	25.74	29.99	13.25
median	9.96	**37.50**	25.03	30.64	9.71

Table 4.1: Out-of-distribution detection on MNIST for every out-of-distribution population defined as the digit left out during training. Digits 1 and 7 are consistently harder to detect, possibly because both consist primarily of a downward stroke and are easy to confuse.

For our out-of-distribution detection problems, we shall report the *area under the PR curve (AUPR)*: this is constructed by varying the threshold t over a range spanning the data, and creating a set of finite points for the PR curve. These points are interpolated to produce a continuous curve (which is an approximation to the true curve), and the area under this curve is estimated to compute our quantity of interest.

We shall also refer to the *skew*, which is the ratio of positive examples to negative examples, and this can serve as a sanity-check baseline, since this is the rate at which we would approximately perform if we randomly guessed labels.

An alternative measure is Receiver Operator Characteristic (ROC), which is similar, but penalizes false positives. Since we assume distributional change is a rare event and would prefer false positives to false negatives in the context of safe AI systems, we use Precision-Recall measures. Moreover, it has also been shown that ROC curves are misleadingly optimistic compared to PR curves for highly skewed datasets [17], and additionally, it is shown that an algorithm that optimizes the area under the ROC curve is not guaranteed to optimize the area under the PR curve.

Evaluation We are evaluating our models as candidates for normality models. The explicit models (VAE, PixelVAE, PixelCNN variants) produce likelihoods, or estimates of likelihoods, which are the most straightfoward measure of normality for such models. For the GAN, we shall use the discriminator output as a surrogate for a measure of normality. Since we are not concerned with the detection module, we shall not tune thresholds, and at evaluation time this is accounted for by AUPR's varying threshold.

OOD category	skew	IAF	PixCNN++	PixelVAE	DC-GAN
airplane	10	5.19	30.39	6.95	10.46
automobile	10	28.83	8.40	14.11	12.00
bird	10	7.48	11.46	9.83	11.38
cat	10	8.57	9.79	9.23	10.47
deer	10	24.60	7.41	11.48	8.37
dog	10	9.71	9.00	8.63	11.32
frog	10	60.30	6.74	17.21	12.73
horse	10	18.78	7.88	10.64	10.57
ship	10	5.36	15.63	6.88	13.14
truck	10	21.96	8.36	11.45	12.50
mean	10	**19.08**	11.51	10.64	11.29
median	10	**14.24**	8.70	10.24	11.35

Table 4.2: Out-of-distribution detection on CIFAR-10 for every out-of-distribution population defined as the object category left out during training. *airplane*, *ship*, and *bird* typically share strong global coherence (an elongated object, sometimes with wing-like structures, typically on a blue background) and are observed to be easily confused for both the VAE and PixelVAE, but PixelCNN appears to be relatively stronger at detecting the difference. We hypothesize this is because PixelCNN is stronger at modelling local precise local details. Interestingly, PixelCNN fails most strongly for *frog*, and we suspect this is because the majority of *frog* images in CIFAR-10 are close-ups, which are locally blurry and thus easily confused with global properties of other categories.

4.3 Results

4.3.1 MNIST with held out digits

In Table 4.1 we report AUPR percentages (a perfect detection would have scores of 100) for comparing normality scores of test set in-distribution vs. out-of-distribution images under the models we are evaluating. Every row corresponds to an out-of-distribution detection experiment where training is done on the training set of MNIST digits but holding out one of the digits, and the out-of-distribution population at test time is defined to be the set of test set images with the held-out-digit. There is considerable variation among the AUPR scores of the different experiments, indicating that certain digits are significantly easier to confuse than others. On average, it would appear the VAE performs the best.

4.3.2 CIFAR-10 with held out classes

In Table 4.2 we show AUPR percentages for CIFAR-10, holding out object categories similarly as before. We replace the simple convolutional VAE with an IAF model with two levels of autoregressive inverse transforms [4], and replace PixelCNN with PixelCNN++, which gives better sample quality. We observe that on average, VAEs are once again the top-performing model, followed by PixelCNN (though PixelCNN actually performs worst according to the median score). The other models are fairly close to baseline performance and the high variance across out-of-distribution experiments means the results are not very interpretable, except perhaps as observations about how internal representations of different models are more/less confused by images with specific objects.

[4] We ran the IAF models for 50 epochs only because it's slower to run, but we visually verified decent sample quality and ~ 3.5 average bits/dim log-likelihood bounds for every experiment.

4.4 Conclusion

This is a hard setup: most often the samples from the distribution with the held-out-class will share a lot of common low-level and even high-level features with the in-distribution training samples; in the absence of labels, the models are given no information about specific discriminative features to look for which would aid detection of novel instances. For images with unseen objects that have similar local features and share similar global coherence with familiar objects (such as a bird and an aeroplane flying in the sky), it is easy for a model with incomplete knowledge about the world to assign a high likelihood to the out-of-distribution example, especially when we are training powerful models on smaller quantities of data.

However, we believe this is a setup that approaches more closely a realistic evaluation of normality models as motivated by the problem of distributional change, in the context of AI safety. Additionally, we also recommend the setup as a possible testing ground for evaluating generative models, with the caveat that this is very goal-oriented. If we wish to implement a generative model where our only concern is that of generating realistic samples, perhaps leaning toward the more "creative" side of imaginative behaviour, we should not care about performance on tasks such as these. If we intend to use them as density models, out-of-distribution detectors, or in applications that stipulate strictly plausible samples [5], this form of evaluation might be relevant. It would help quantify a generative model's propensity to "over-generalize" with potentially undesired outcomes.

Acknowledgements

We would like to thank Ishaan Gulrajani and Daniel Hendrycks for useful discussions. We would also like to thank the developers of Theano [81] and Blocks and Fuel [50], and acknowledge the support of the following agencies for research funding and computing support: Ubisoft, Nuance Foundation, NSERC, Calcul Quebec, Compute Canada, CIFAR, MEC Project TRA2014-57088-C2-1-R, SGR project 2014-SGR-1506 and TECNIOspring-FP7-ACCIÓ grant.

[5] We mean 'strictly plausible' in the sense that samples with objects that differ significantly from ones in the training distribution are not generated with significantly high likelihood.

Chapter 5

Conclusion

Unlabelled data is typically much more readily available and in significantly larger quantities than labelled data, and can potentially convey an enormous amount of implicit information. In this thesis, we have observed that generative models are powerful tools for learning highly complex distributions over high-dimensional data in an unsupervised manner, so they are well-suited to exploit these vast quantities of unlabelled data. As we discussed in chapter 1, with the ability to produce meaningful samples, generative models carry in them an implicit understanding of the world, and being able to learn and represent knowledge in an unsupervised way is a core component of efficient machine intelligence.

In chapter 2, we introduced a latent variable model with an autoregressive decoder. We qualitatively demonstrated the ability of this model to learn a semantically hierarchical latent space. We also showed that this latent space learns more global features, and with greater compression compared to a standard VAE. This indicates potential usefulness in downstream representation learning tasks, which will be explored in future work.

In chapter 3, we investigated some of the problems arising from weight clipping in WGANs. We presented an alternative by introducing a penalty term in the critic loss, which led to clear benefits in modelling performance, training stability, and promising results with modelling discrete data. With this contribution, we can now focus efforts upon scaling up image modelling to higher resolutions, and exploring the use of GANs in discrete output spaces.

In chapter 4, we talked about the possibility of AI systems behaving in ways that are misaligned with human interests, when they are deployed in the real world. For example, a robot trained to clear the floor of trash might throw away a valuable object that was accidentally dropped. One way to reduce such undesired behaviour would be letting our AI systems know when they face novel situations. We have approached a possible direction for detecting novel instances using generative models, but without particularly promising results. With the current models that we have, perhaps modelling the density of the training distribution is not enough; we might need to explicitly model *surprise at novelty*. This might call for some supervision in training, where we shall provide a reward signal for surprise.

A line of future work, building upon the work presented in this thesis, would be focussing on the current challenge of generating video. For this task, apart from modelling spatial structure, we shall need to represent temporal structure as well. Recurrent neural networks lend themselves naturally to the task, but have been hard to train. In the spirit of our work presented in chapter 2, a hierarchical decomposition across the temporal axis might be useful, and will be considered for future work. Exploring the use of improved training techniques, such as the one presented in chapter 3, is also a possible direction.

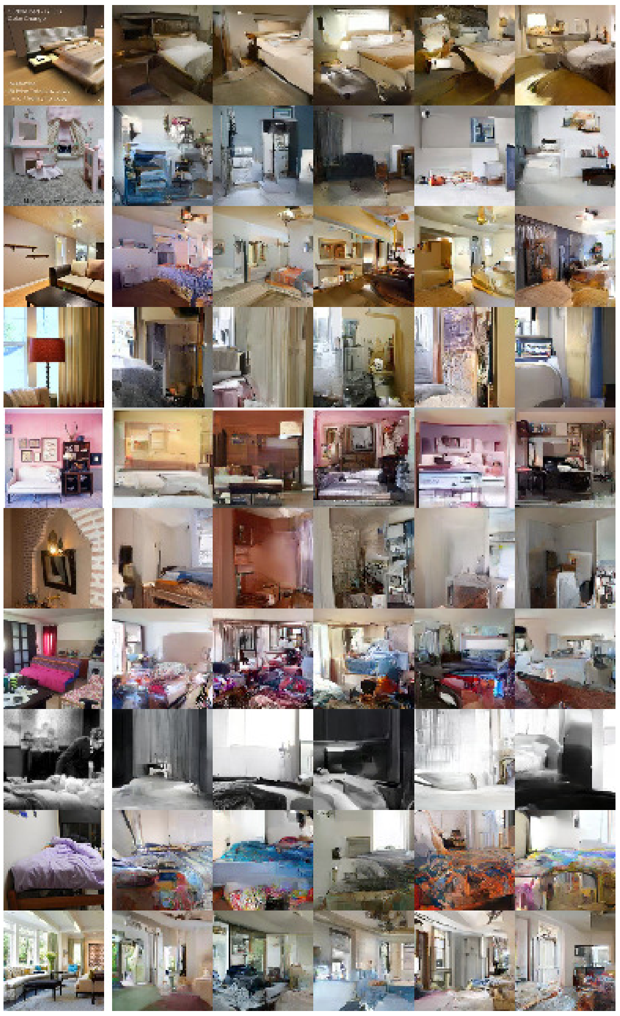
Another line of future work follows from the idea that a generative model capable of capturing greater detail is one that can potentially incorporate a more precise understanding of the world. This motivates scaling up generative modelling to images of higher resolution, which comes with the need for increased modelling capacity as well as efficient sampling mechanisms. We hypothesize that the current inadequacy of generative models to serve as accurate models of normality might be resolved with being able to efficiently model images of greater precision.

Artificial intelligence is well on the way to pervade most, if not all, aspects of human life. Generative models, in particular, can augment our artistic and creative endeavours, in addition to other applications, some of which we have discussed. Research and development in the near future are expected to enhance the presence of machine intelligence in our lives, and powerful generative models would help augment the machines in our lives with cogent reasoning skills and creativity.

Appendix A

Appendix for PixelVAE: A latent variable model for natural images

A.0.1 LSUN bedrooms and 64×64 ImageNet reconstructions


(a)

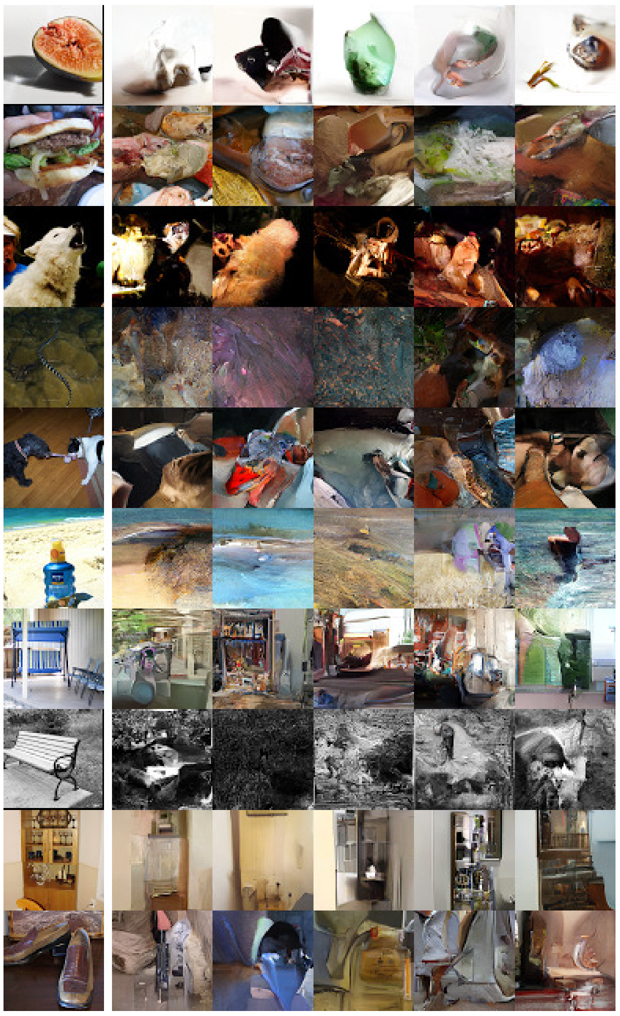

(b)

Figure A.1: Reconstructions for (a) LSUN Bedrooms and (b) 64×64 ImageNet. Left-most columns are images from the test set, and the following 5 columns are top-down generations from the highest level of latent variables. We see that the reconstructions capture high-level semantic properties of the original images while varying in most of the details. We also visualized similar reconstructions by generations from the lower level of latent variables, and in this case the reconstructions were visually indistinguishable from the original images.

A.0.2 MNIST samples

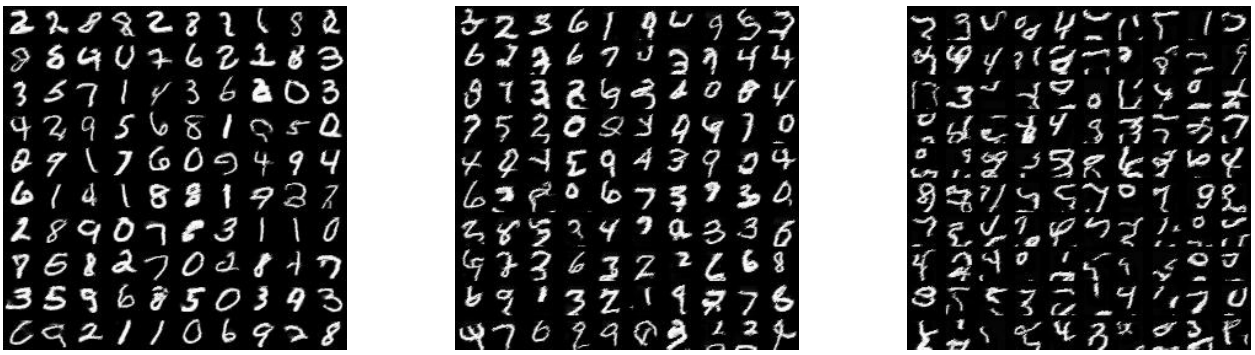


Figure A.2: Samples from a PixelVAE with a receptive field of 7 pixels (left), a PixelCNN with an 11-pixel receptive field (middle; roughly the same computational complexity as the PixelVAE), and a PixelCNN with a 7-pixel receptive field (right).

A.0.3 MNIST reconstructions

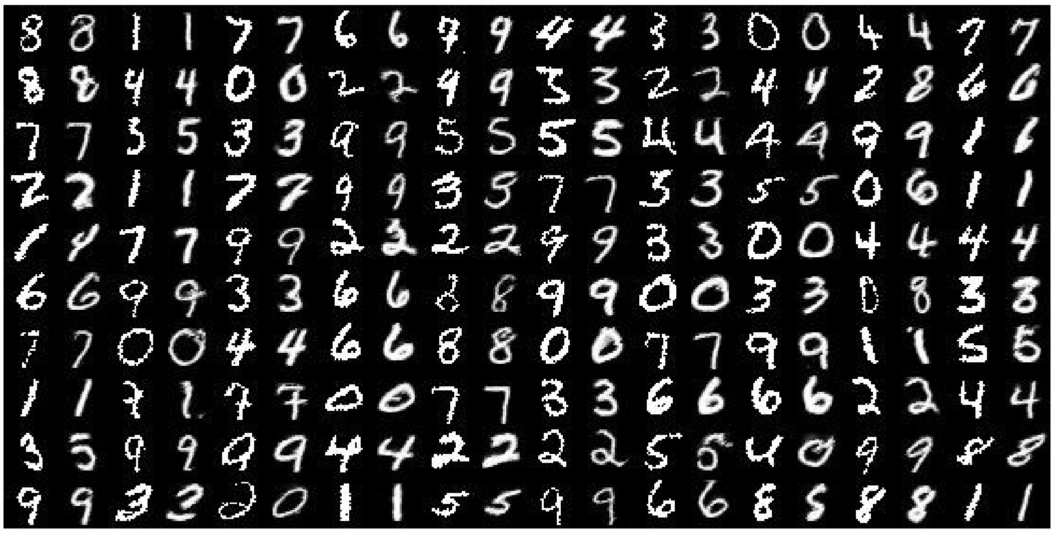


Figure A.3: Reconstructions from the MNIST test set. Alternate columns are original (left) and reconstructed images (right).

A.0.4 More samples for hierarchical latent space visualizations

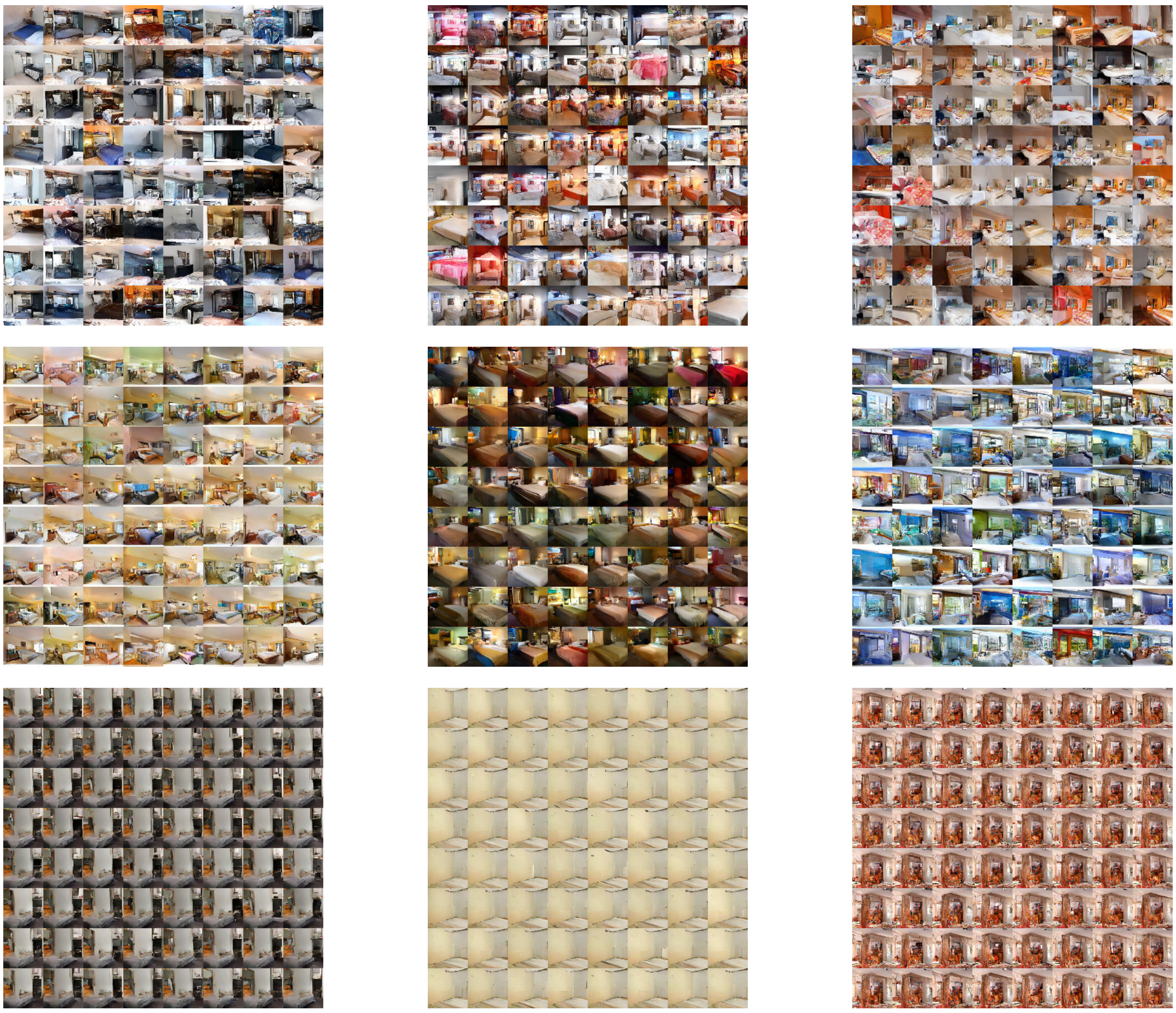


Figure A.4: More examples for visualizations of the variation in image features captured at different levels of stochasticity. Holding the other levels constant, we vary only the top-level sampling noise *(top)*, only the middle-level noise *(middle)*, and only the bottom (pixel-level) noise *(bottom)*.

A.0.5 Model architecture

MNIST

For our quantitative MNIST experiments, the architectures of our encoder and decoder are as follows. Unless otherwise specified, all convolutional layers use ReLU nonlinearity. We also make an open-source implementation of this model available at `https://github.com/igul222/PixelVAE`.

Encoder $x \to (\mu, \sigma)$			
	Kernel size	Stride	Output channels
Convolution	3x3	1	32
Convolution	3x3	2	32
Convolution	3x3	1	32
Convolution	3x3	2	64
Pad 7×7 feature maps to 8×8			
Convolution	3x3	1	64
Convolution	3x3	2	64
Convolution	3x3	1	64
Convolution	3x3	1	64
Convolution	3x3	1	64
Flatten			
Linear	-	-	2×latent dimensionality

Decoder $z \to x$			
	Kernel size	Stride	Output channels
Linear	-	-	4×4×64
Reshape to (64, 4, 4)			
Convolution	3x3	1	64
Convolution	3x3	1	64
Transposed convolution	3x3	2	64
Convolution	3x3	1	64
Crop 8×8 feature maps to 7×7			
Transposed convolution	3x3	2	32
Convolution	3x3	1	32
Transposed convolution	3x3	2	32
Convolution	3x3	1	32
PixelCNN gated residual block	7x7	1	32
PixelCNN gated residual block(s)	[5x5] $\times N$	1	32
PixelCNN gated convolution	1x1	1	32
PixelCNN gated convolution	1x1	1	32
Convolution	1x1	1	1

LSUN bedrooms and 64×64 ImageNet

The LSUN and ImageNet models use the same architecture: all encoders and decoders are residual networks; we use pre-activation residual blocks with two 3×3 convolutional layers each and ELU nonlinearity. Some residual blocks perform downsampling, using a 2×2 stride in the second convolutional layer, or upsampling, using subpixel convolution in the first convolutional layer. Weight normalization is used in masked convolutional layers; in all other layers, batch normalization is used. We optimize using Adam with learning rate 1e-3. Training proceeds for 400K iterations using batch size 48.

For further architectural details, please refer to our open-source implementation at `https://github.com/igul222/PixelVAE`.

Bottom-level Encoder $x \to h_1$			
	Kernel size	Resample	Output channels
Embedding	-	-	48
Convolution	1x1	-	192
Residual block	[3x3] $\times$ 2	-	192
Residual block	[3x3] $\times$ 2	Down $\times$2	256
Residual block	[3x3] $\times$ 2	-	256
Residual block	[3x3] $\times$ 2	Down $\times$2	512
Residual block	[3x3] $\times$ 2	-	512
Residual block	[3x3] $\times$ 2	-	512
Residual block	[3x3] $\times$ 2	-	512

Bottom-level Decoder $z_1 \to x$			
	Kernel size	Resample	Output channels
Convolution	1x1	1	512
Residual block	[3x3] $\times$ 2	-	512
Residual block	[3x3] $\times$ 2	-	512
Residual block	[3x3] $\times$ 2	-	512
Residual block	[3x3] $\times$ 2	Up $\times$2	256
Residual block	[3x3] $\times$ 2	-	256
Residual block	[3x3] $\times$ 2	Up $\times$2	192
Residual block	[3x3] $\times$ 2	-	192
Embedding	-	-	48
PixelCNN gated residual block	[3x3] $\times$ 2	-	384
PixelCNN gated residual block	[3x3] $\times$ 2	-	384
PixelCNN gated residual block	[3x3] $\times$ 2	-	384

Top-level Encoder $h_1 \to h_2$			
	Kernel size	Resample	Output channels
Residual block	[3x3] $\times$ 2	-	512
Residual block	[3x3] $\times$ 2	-	512
Residual block	[3x3] $\times$ 2	Down $\times$2	512
Residual block	[3x3] $\times$ 2	-	512
Residual block	[3x3] $\times$ 2	-	512
Residual block	[3x3] $\times$ 2	Down $\times$2	512
Residual block	[3x3] $\times$ 2	-	512
Residual block	[3x3] $\times$ 2	-	512

Top-level Decoder $z_2 \rightarrow z_1$			
	Kernel size	Resample	Output channels
Linear	-	-	4×4×512
Reshape to (512, 4, 4)			
Residual block	[3x3] × 2	-	512
Residual block	[3x3] × 2	-	512
Residual block	[3x3] × 2	Up ×2	512
Residual block	[3x3] × 2	-	512
Residual block	[3x3] × 2	-	512
Residual block	[3x3] × 2	Up ×2	512
Residual block	[3x3] × 2	-	512
Residual block	[3x3] × 2	-	512
PixelCNN convolution	5x5	-	512
PixelCNN gated residual block	[3x3] × 2	-	512
PixelCNN gated residual block	[3x3] × 2	-	512
PixelCNN gated residual block	[3x3] × 2	-	512
Convolution	1x1	-	256

Appendix B

Appendix for Improved training of Wasserstein GANs

B.1 Proof about the optimal WGAN critic

Lemma 1. *Let $\mathbb{P}_r$ and $\mathbb{P}_g$ be two distributions in $\mathcal{X}$, a compact metric space. Then, there is a 1-Lipschitz function f^* which is the optimal solution of*

$$\max_{\|f\|_L \leq 1} \underset{y\sim\mathbb{P}_r}{\mathbb{E}}[f(y)] - \underset{x\sim\mathbb{P}_g}{\mathbb{E}}[f(x)]$$

Let π be the optimal coupling between $\mathbb{P}_r$ and $\mathbb{P}_g$, defined as the minimizer of:

$$W(\mathbb{P}_r, \mathbb{P}_g) = \inf_{\pi\in\Pi(\mathbb{P}_r,\mathbb{P}_g)} \underset{(x,y)\sim\pi}{\mathbb{E}}[\|x-y\|]$$

Where $\Pi(\mathbb{P}_r, \mathbb{P}_g)$ is the set of joint distributions $\pi(x, y)$ whose marginals are $\mathbb{P}_r$ and $\mathbb{P}_g$, respectively. Then, if f^* is differentiable [1], *$\pi(x = y) = 0$*[2] *and $x_t = tx + (1-t)y$ with $0 \leq t \leq 1$,*

$$\mathbb{P}_{(x,y)\sim\pi}\left[\nabla f^*(x_t) = \frac{y - x_t}{\|y - x_t\|}\right] = 1$$

This, in particular, implies the norms of the gradients are 1.

Proof. Since $\mathcal{X}$ is a compact space, by Theorem 5.10 of [84], part (iii), we know that there is an optimal f^*. By Theorem 5.10 of [84], part (ii) we know that if π is an optimal coupling,

$$\mathbb{P}_{(x,y)\sim\pi}[f^*(y) - f^*(x) = \|y - x\|] = 1$$

Let (x, y) be such that $f^*(y) - f^*(x) = \|y - x\|$. We can safely assume that $x \neq y$ as well, since this happens under π with probability 1. Let $\psi(t) = f^*(x_t) - f^*(x)$. We claim that $\psi(t) = \|x_t - x\| = t\|y - x\|$.
Let $t, t' \in [0, 1]$, then

$$\begin{aligned}|\psi(t) - \psi(t')| &= \|f^*(x_t) - f^*(x_{t'})\| \\ &\leq \|x_t - x_{t'}\| \\ &= |t - t'|\|x - y\|\end{aligned}$$

[1] We can actually assume much less, and talk only about directional derivatives on the direction of the line; which we show in the proof always exist. This would imply that in every point where f^* is differentiable (and thus we can take gradients in a neural network setting) the statement holds.

[2] This assumption is in order to exclude the case when the matching point of sample x is x itself. It is satisfied in the case that $\mathbb{P}_r$ and $\mathbb{P}_g$ have supports that intersect in a set of measure 0, such as when they are supported by two low dimensional manifolds that don't perfectly align [2].

Therefore, ψ is $\|x-y\|$-Lipschitz. This in turn implies

$$\begin{aligned}\psi(1)-\psi(0) &= \psi(1)-\psi(t)+\psi(t)-\psi(0)\\ &\le (1-t)\|x-y\|+\psi(t)-\psi(0)\\ &\le (1-t)\|x-y\|+t\|x-y\|\\ &= \|x-y\|\end{aligned}$$

However, $\psi(1)-\psi(0)=f^*(y)-f^*(x)=\|y-x\|$ so the inequalities have to actually be equalities. In particular, $\psi(t)-\psi(0)=t\|x-y\|$, and $\psi(0)=f^*(x)-f^*(x)=0$. Therefore, $\psi(t)=t\|x-y\|$ and we finish our claim.

Let

$$\begin{aligned}v &= \frac{y-x_t}{\|y-x_t\|}\\ &= \frac{y-((1-t)x-ty)}{\|y-((1-t)x-ty)\|}\\ &= \frac{(1-t)(y-x)}{\|(1-t)\|y-x\|}\\ &= \frac{y-x}{\|y-x\|}\end{aligned}$$

Now we know that $f^*(x_t)-f^*(x)=\psi(t)=t\|x-y\|$, so $f^*(x_t)=f^*(x)+t\|x-y\|$. Then, we have the partial derivative

$$\begin{aligned}\frac{\partial}{\partial v}f^*(x_t) &= \lim_{h\to 0}\frac{f^*(x_t+hv)-f^*(x_t)}{h}\\ &= \lim_{h\to 0}\frac{f^*\left(x+t(y-x)+\frac{h}{\|y-x\|}(y-x)\right)-f^*(x_t)}{h}\\ &= \lim_{h\to 0}\frac{f^*\left(x_{t+\frac{h}{\|y-x\|}}\right)-f^*(x_t)}{h}\\ &= \lim_{h\to 0}\frac{f^*(x)+\left(t+\frac{h}{\|y-x\|}\right)\|x-y\|-(f^*(x)+t\|x-y\|)}{h}\\ &= \lim_{h\to 0}\frac{h}{h}\\ &= 1\end{aligned}$$

If f^* is differentiable at x_t, we know that $\|\nabla f^*(x_t)\|\le 1$ since it is a 1-Lipschitz function. Therefore, by simple Pythagoras and using that v is a unit vector

$$\begin{aligned}1 &\le \|\nabla f^*(x)\|^2\\ &= \langle v,\nabla f^*(x_t)\rangle^2+\|\nabla f^*(x_t)-\langle v,\nabla f^*(x_t)\rangle v\|^2\\ &= |\frac{\partial}{\partial v}f^*(x_t)|^2+\|\nabla f^*(x_t)-v\frac{\partial}{\partial v}f^*(x_t)\|^2\\ &= 1+\|\nabla f^*(x_t)-v\|^2\\ &\le 1\end{aligned}$$

The fact that both extremes of the inequality coincide means that it was all an equality and $1=1+\|\nabla f^*(x_t)-v\|^2$ so $\|\nabla f^*(x_t)-v\|=0$ and therefore $\nabla f^*(x_t)=v$. This shows that $\nabla f^*(x_t)=\frac{y-x_t}{\|y-x_t\|}$.

To conclude, we showed that if (x, y) have the property that $f^*(y) - f^*(x) = \|y - x\|$, then $\nabla f^*(x_t) = \frac{y - x_t}{\|y - x_t\|}$. Since this happens with probability 1 under π, we know that

$$\mathbb{P}_{(x,y)\sim\pi}\left[\nabla f^*(x_t) = \frac{y - x_t}{\|y - x_t\|}\right] = 1$$

and we finished the proof. $\square$

B.2 Hyperparameters used for stability experiments

- WGAN with gradient penalty: Adam ($\alpha = .0001, \beta_1 = .5, \beta_2 = .9$)
- WGAN with weight clipping: RMSProp ($\alpha = .00005$)
- DCGAN: Adam ($\alpha = .0002, \beta_1 = .5$)
- LSGAN: RMSProp ($\alpha = .0001$) [chosen by search over $\alpha = .001, .0002, .0001$]

B.3 CIFAR-10 ResNet architecture

The generator and critic are residual networks; we use pre-activation residual blocks with two 3×3 convolutional layers each and ReLU nonlinearity. Some residual blocks perform downsampling (in the critic) using mean pooling after the second convolution, or nearest-neighbor upsampling (in the generator) before the second convolution. We use batch normalization in the generator but not the critic. We optimize using Adam with learning rate 2×10^{-4}, decayed linearly to 0 over 100K generator iterations, and batch size 64.

For further architectural details, please refer to our open-source implementation.

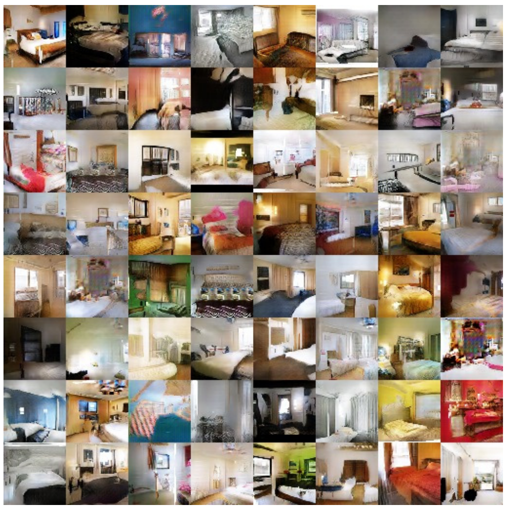
Generator $G(z)$

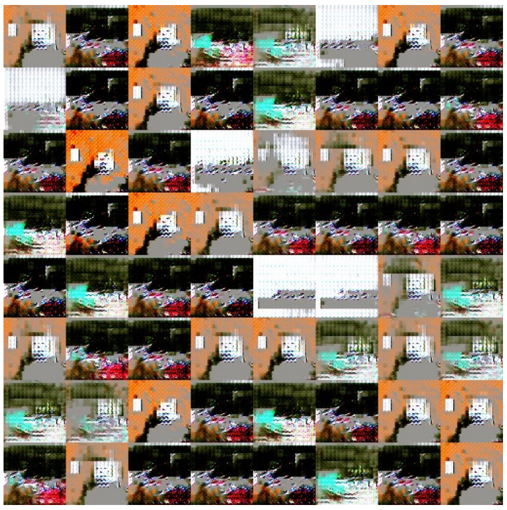
	Kernel size	Resample	Output shape
z	-	-	128
Linear	-	-	$128 \times 4 \times 4$
Residual block	[3×3] × 2	Up	$128 \times 8 \times 8$
Residual block	[3×3] × 2	Up	$128 \times 16 \times 16$
Residual block	[3×3] × 2	Up	$128 \times 32 \times 32$
Conv, *tanh*	3×3	-	$3 \times 32 \times 32$

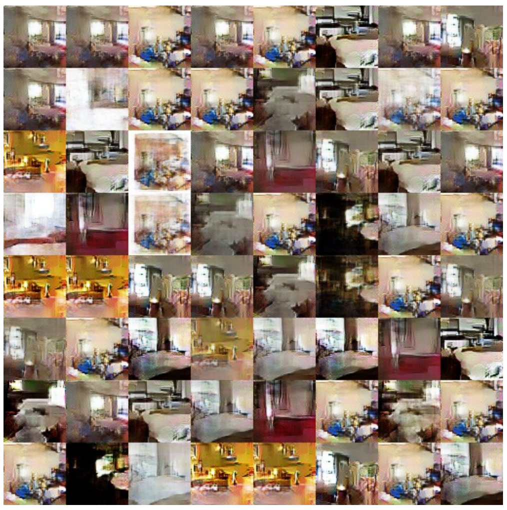
Critic $D(x)$

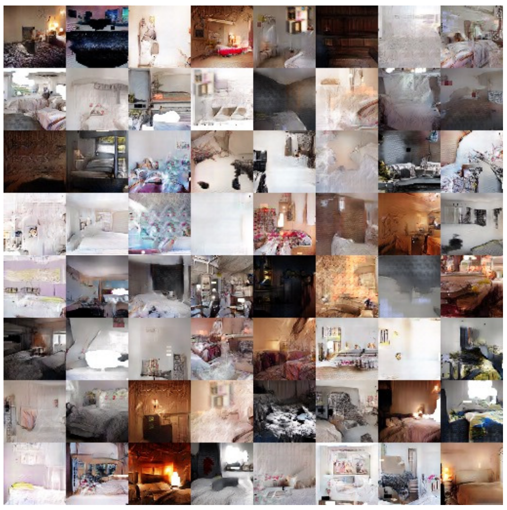
	Kernel size	Resample	Output shape
Residual block	[3×3] × 2	Down	$128 \times 16 \times 16$
Residual block	[3×3] × 2	Down	$128 \times 8 \times 8$
Residual block	[3×3] × 2	-	$128 \times 8 \times 8$
Residual block	[3×3] × 2	-	$128 \times 8 \times 8$
ReLU, mean pool	-	-	128
Linear	-	-	1

B.4 More LSUN samples

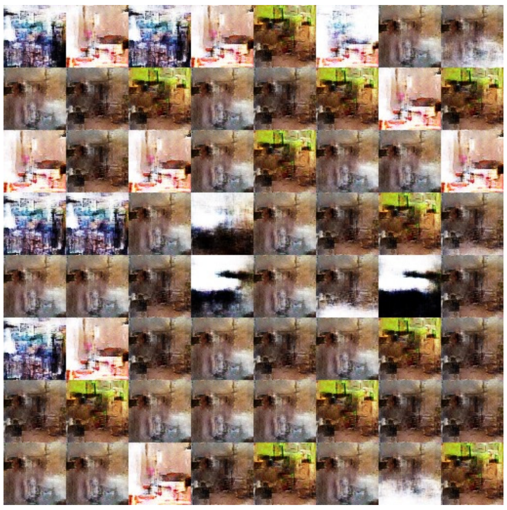

Method: DCGAN
G: DCGAN, D: DCGAN

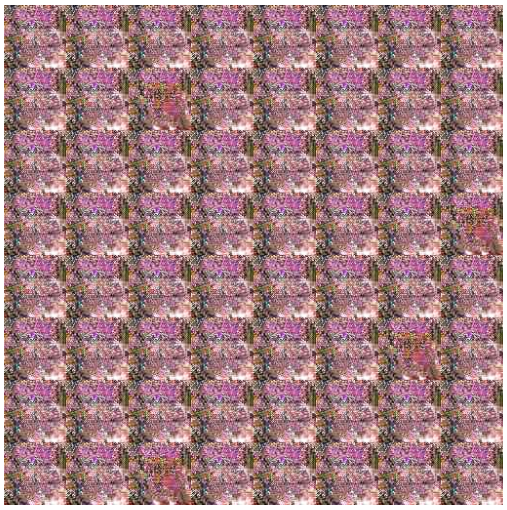

Method: DCGAN
G: No BN and const. filter count

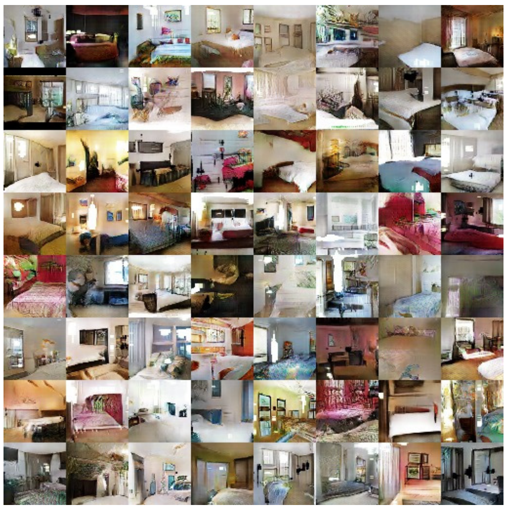

Method: DCGAN
G: 4-layer 512-dim ReLU MLP

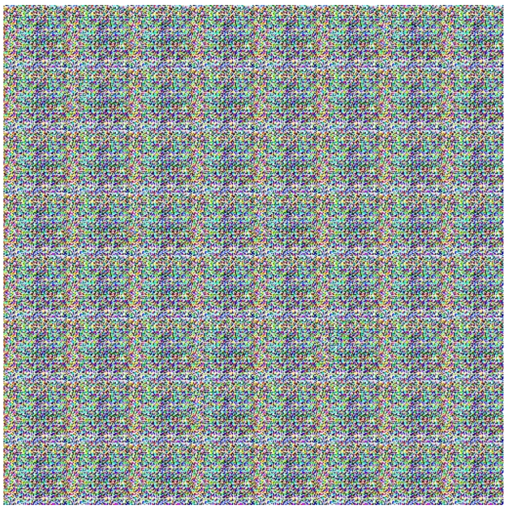

Method: DCGAN
No normalization in either G or D

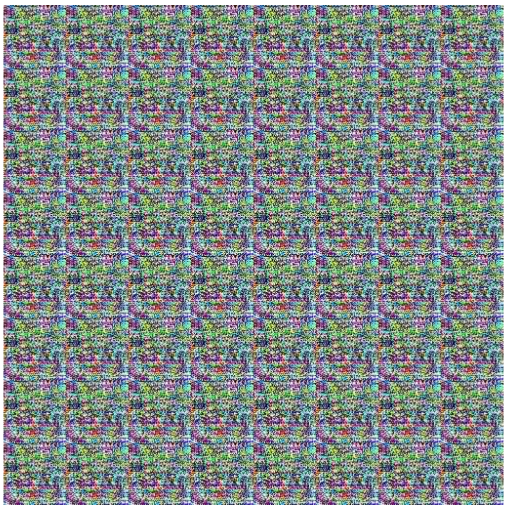
Method: DCGAN
Gated multiplicative nonlinearities

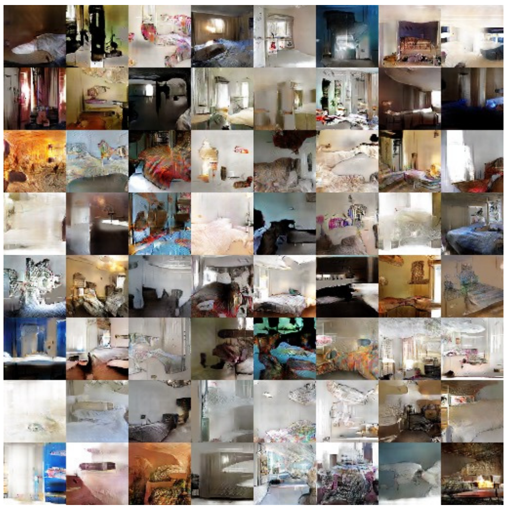

Method: DCGAN
tanh nonlinearities

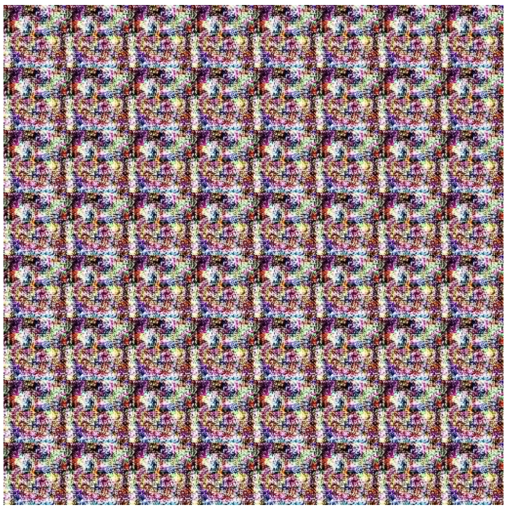

Method: DCGAN
101-layer ResNet G and D

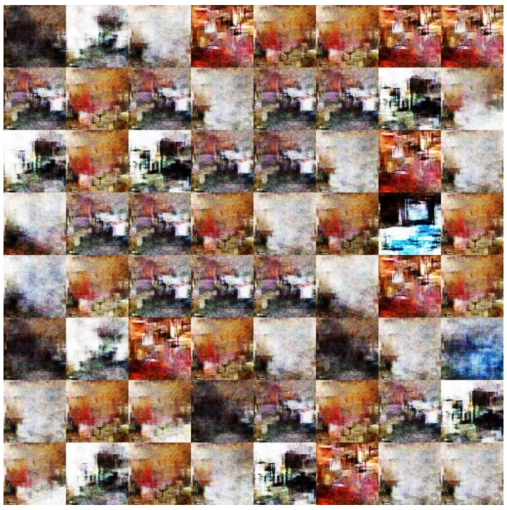

Method: LSGAN
G: DCGAN, D: DCGAN

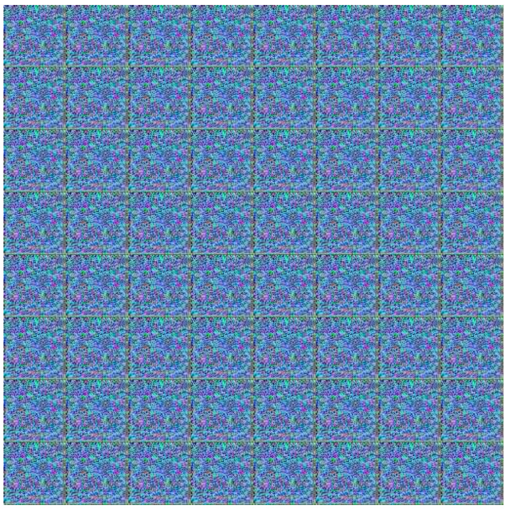

Method: LSGAN
G: No BN and const. filter count

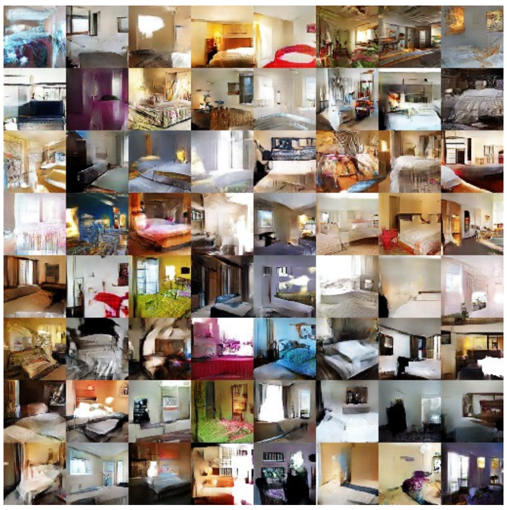

Method: LSGAN
G: 4-layer 512-dim ReLU MLP

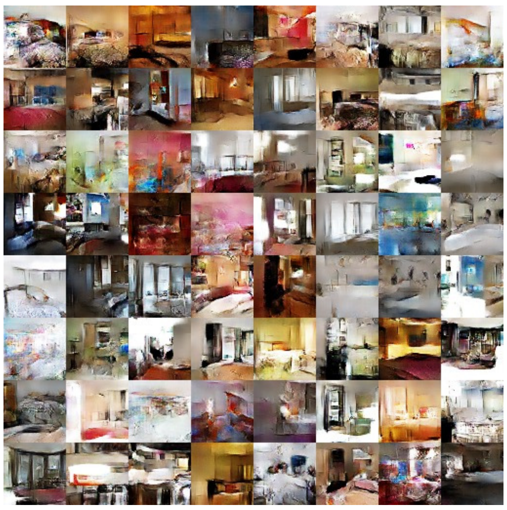

Method: LSGAN
No normalization in either G or D


Method: LSGAN
Gated multiplicative nonlinearities

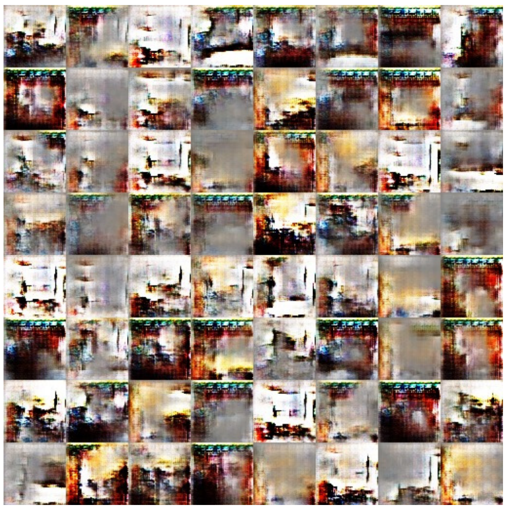

Method: LSGAN
tanh nonlinearities


Method: LSGAN
101-layer ResNet G and D


Method: WGAN with clipping
G: DCGAN, D: DCGAN


Method: WGAN with clipping
G: No BN and const. filter count

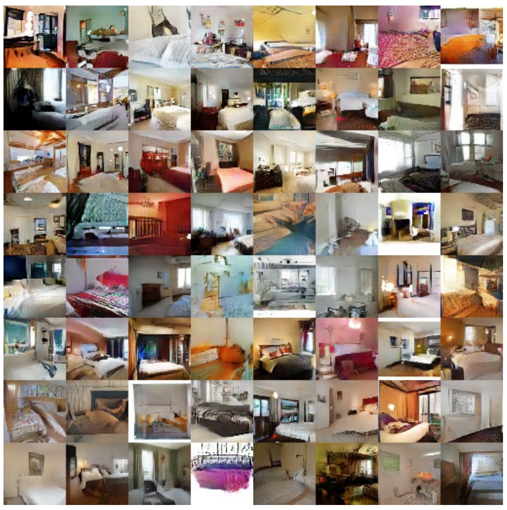
Method: WGAN with clipping
G: 4-layer 512-dim ReLU MLP

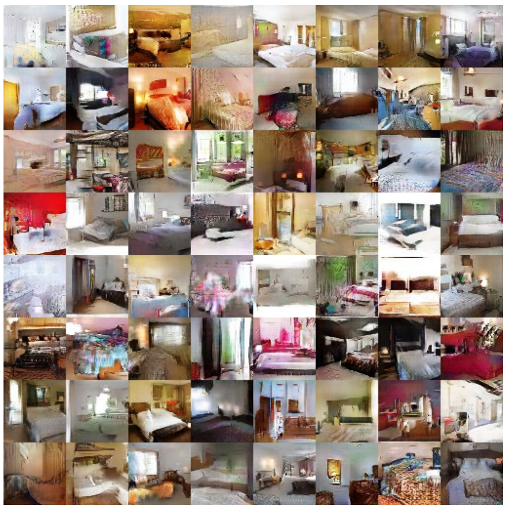

Method: WGAN with clipping
No normalization in either G or D

Method: WGAN with clipping
Gated multiplicative nonlinearities

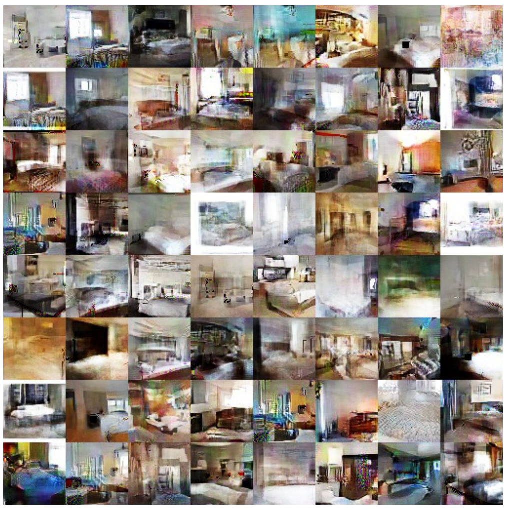
Method: WGAN with clipping
tanh nonlinearities

Method: WGAN with clipping
101-layer ResNet G and D

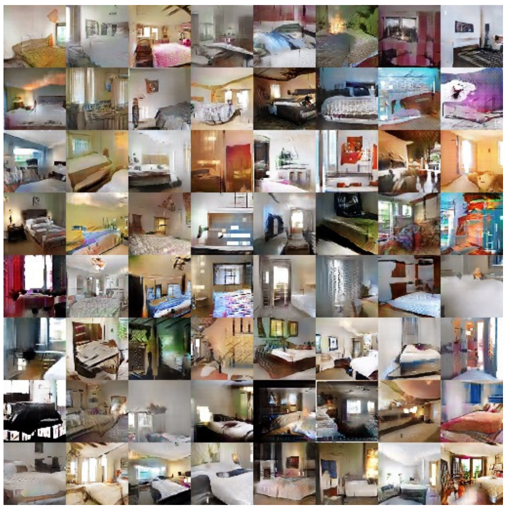

Method: WGAN-GP (ours)
G: DCGAN, D: DCGAN



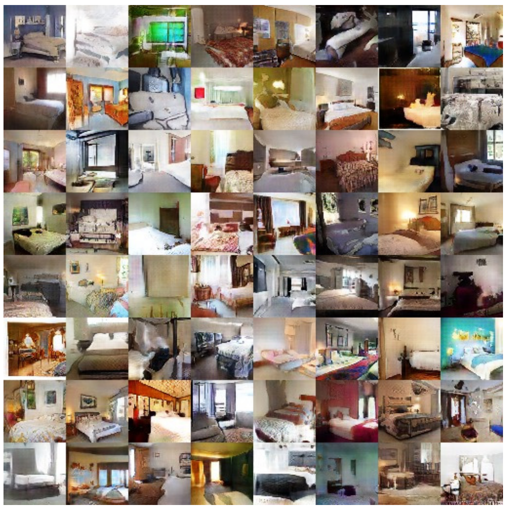
Method: WGAN-GP (ours)
G: No BN and const. filter count



Method: WGAN-GP (ours)
G: 4-layer 512-dim ReLU MLP

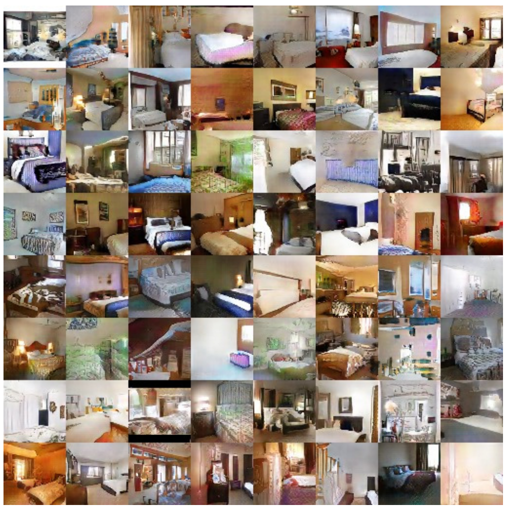


Method: WGAN-GP (ours)
No normalization in either G or D



Method: WGAN-GP (ours)
Gated multiplicative nonlinearities

Method: WGAN-GP (ours)
tanh nonlinearities



Method: WGAN-GP (ours)
101-layer ResNet G and D

Bibliography

[1] Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. "Concrete Problems in AI Safety". In: *CoRR* abs/1606.06565 (2016). URL: http://arxiv.org/abs/1606.06565.

[2] Martin Arjovsky and Léon Bottou. "Towards Principled Methods for Training Generative Adversarial Networks". In: *ICLR* (2017).

[3] Martin Arjovsky, Soumith Chintala, and Léon Bottou. "Wasserstein Generative Adversarial Networks". In: *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*. 2017, pp. 214–223.

[4] Martin Arjovsky, Amar Shah, and Yoshua Bengio. "Unitary Evolution Recurrent Neural Networks". In: *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*. ICML'16. New York, NY, USA: JMLR.org, 2016, pp. 1120–1128. URL: http://dl.acm.org/citation.cfm?id=3045390.3045509.

[5] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. "Layer normalization". In: *arXiv preprint arXiv:1607.06450* (2016).

[6] David Berthelot, Tom Schumm, and Luke Metz. "BEGAN: Boundary Equilibrium Generative Adversarial Networks". In: *arXiv preprint arXiv:1703.10717* (2017).

[7] Samuel R Bowman, Luke Vilnis, Oriol Vinyals, Andrew M Dai, Rafal Jozefowicz, and Samy Bengio. "Generating sentences from a continuous space". In: 2016.

[8] Yuri Burda, Roger Grosse, and Ruslan Salakhutdinov. "Importance Weighted Autoencoders". In: *arXiv preprint arXiv:1509.00519* (2015).

[9] Kenneth P. Burnham and David R. Anderson. "Model selection and Multi-model inference, 2nd ed." In: *A Practical information-theoretic approach. Springer-Verlag* (2003), p. 78.

[10] Tong Che, Yanran Li, Ruixiang Zhang, R Devon Hjelm, Wenjie Li, Yangqiu Song, and Yoshua Bengio. "Maximum-Likelihood Augmented Discrete Generative Adversarial Networks". In: *arXiv preprint arXiv:1702.07983* (2017).

[11] Ciprian Chelba, Tomas Mikolov, Mike Schuster, Qi Ge, Thorsten Brants, Phillipp Koehn, and Tony Robinson. "One billion word benchmark for measuring progress in statistical language modeling". In: *arXiv preprint arXiv:1312.3005* (2013).

[12] Xi Chen, Diederik P Kingma, Tim Salimans, Yan Duan, Prafulla Dhariwal, John Schulman, Ilya Sutskever, and Pieter Abbeel. "Variational Lossy Autoencoder". In: *arXiv.org* (Nov. 2016). arXiv: 1611.02731v1 [cs.LG].

[13] Soumith Chintala, Emily Denton, Martin Arjovsky, and Michael Mathieu. *How to train a GAN? Tips and tricks to make GANs work*. https://github.com/soumith/ganhacks. Accessed: 2017-08-08.

[14] Kyunghyun Cho, Bart van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. "On the properties of neural machine translation: Encoder-decoder approaches". In: *Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation (SSST-8), 2014*. 2014.

[15] Junyoung Chung, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. "Empirical evaluation of gated recurrent neural networks on sequence modeling". In: *arXiv preprint arXiv:1412.3555* (2014).

[16] Zihang Dai, Amjad Almahairi, Philip Bachman, Eduard Hovy, and Aaron Courville. "Calibrating energy-based generative adversarial networks". In: *arXiv preprint arXiv:1702.01691* (2017).

[17] Jesse Davis and Mark Goadrich. "The Relationship Between Precision-Recall and ROC Curves". In: *Proceedings of the 23rd International Conference on Machine Learning*. Pittsburgh, Pennsylvania, USA, 2006, pp. 233–240.

[18] Peter J. Diggle and Richard J. Gratton. "Monte Carlo Methods of Inference for Implicit Statistical Models". In: *Journal of the Royal Statistical Society* 46.2 (1984), pp. 193–227.

[19] Jeff Donahue, Philipp Krähenbühl, and Trevor Darrell. "Adversarial Feature Learning". In: *CoRR* abs/1605.09782 (2016). URL: http://arxiv.org/abs/1605.09782.

[20] Vincent Dumoulin, Mohamed Ishmael Diwan Belghazi, Ben Poole, Alex Lamb, Martin Arjovsky, Olivier Mastropietro, and Aaron Courville. "Adversarially Learned Inference". In: 2017. URL: https://openreview.net/forum?id=B1ElR4cgg.

[21] Luba Elliot and Peter Zhegin. *How to take AI far beyond gaming*. https://venturebeat.com/2017/03/14/ho to-take-ai-far-beyond-gaming/. Accessed: 29 Nov 2017. 2017.

[22] Matthieu Germain, Karol Gregor, Iain Murray, and Hugo Larochelle. "Made: Masked autoencoder for distribution estimation". In: *CoRR* abs/1502.03509 (2015). URL: https://arxiv.org/abs/1502.03509.

[23] 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*. 2014, pp. 2672–2680.

[24] Karol Gregor, Frederic Besse, Danilo Jimenez Rezende, Ivo Danihelka, and Daan Wierstra. "Towards Conceptual Compression". In: *arXiv.org* (Apr. 2016). arXiv: 1604.08772v1 [stat.ML].

[25] Rafael Gómez-Bombarelli, David Duvenaud, José Miguel Hernández-Lobato, Jorge Aguilera-Iparraguirre, Timothy D Hirzel, Ryan P Adams, and Alán Aspuru-Guzik. In: *arXiv preprint arXiv:1610.02415* (2016).

[26] J. Hadamard. *Mémoire sur le problème d'analyse relatif à l'équilibre des plaques élastiques encastrées*. Mémoires présentés par divers savants à l'Académie des sciences de l'Institut de France: Éxtrait. Imprimerie nationale, 1908. URL: http://books.google.com.au/books?id=BTEPAAAAIAAJ.

[27] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep Residual Learning for Image Recognition". In: *Computer Vision and Pattern Recognition (CVPR), 2016 IEEE Conference on*. 2016.

[28] Irina Higgins, Loïc Matthey, Xavier Glorot, Arka Pal, Benigno Uria, Charles Blundell, Shakir Mohamed, and Alexander Lerchner. "Early Visual Concept Learning with Unsupervised Deep Learning". In: *CoRR* abs/1606.05579 (2016). URL: http://arxiv.org/abs/1606.05579.

[29] R Devon Hjelm, Athul Paul Jacob, Tong Che, Kyunghyun Cho, and Yoshua Bengio. "Boundary-Seeking Generative Adversarial Networks". In: *arXiv preprint arXiv:1702.08431* (2017).

[30] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural Comput.* 9.8 (Nov. 1997), pp. 1735–1780. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735. URL: http://dx.doi.org/10.1162/neco.1997.9.8.1735.

[31] Xun Huang, Yixuan Li, Omid Poursaeed, John Hopcroft, and Serge Belongie. “Stacked Generative Adversarial Networks”. In: *arXiv preprint arXiv:1612.04357* (2016).

[32] 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, ICML 2015, Lille, France, 6-11 July 2015*. 2015, pp. 448–456.

[33] Viren Jain and H. Sebastian Seung. “Natural Image Denoising with Convolutional Networks”. In: *Proceedings of the 21st International Conference on Neural Information Processing Systems*. NIPS'08. Vancouver, British Columbia, Canada: Curran Associates Inc., 2008, pp. 769–776. ISBN: 978-1-6056-0-949-2. URL: http://dl.acm.org/citation.cfm?id=2981780.2981876.

[34] Eric Jang, Shixiang Gu, and Ben Poole. “Categorical Reparameterization with Gumbel-Softmax”. In: *arXiv preprint arXiv:1611.01144* (2016).

[35] Diederik P. Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*. 2014.

[36] Diederik P. Kingma, Tim Salimans, and Max Welling. “Improving Variational Inference with Inverse Autoregressive Flow”. In: *CoRR* abs/1606.04934 (2016).

[37] Diederik P. Kingma and Max Welling. “Auto-Encoding Variational Bayes”. In: *International Conference on Learning Representations (ICLR)* (2014).

[38] Diederik P Kingma, Tim Salimans, Rafal Jozefowicz, Xi Chen, Ilya Sutskever, and Max Welling. “Improved Variational Inference with Inverse Autoregressive Flow”. In: *Advances in Neural Information Processing Systems 29*. 2016, pp. 4743–4751.

[39] Diederik P. Kingma, Shakir Mohamed, Danilo J. Rezende, and Max Welling. “Semi-supervised Learning with Deep Generative Models”. In: *Advances in Neural Information Processing Systems 27*. Ed. by Z. Ghahramani, M. Welling, C. Cortes, N.d. Lawrence, and K.q. Weinberger. Curran Associates, Inc., 2014, pp. 3581–3589. URL: http://papers.nips.cc/paper/5352-semi-supervised-learning-with-deep-generative-models.pdf.

[40] Alex Krizhevsky. “Learning multiple layers of features from tiny images”. In: (2009).

[41] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. “Backpropagation applied to handwritten zip code recognition”. In: *Neural computation* 1.4 (1989), pp. 541–551.

[42] Yann Lecun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE*. 1998, pp. 2278–2324.

[43] C. Ledig, L. Theis, F. Huszar, J. Caballero, A. Aitken, A. Tejani, J. Totz, Z. Wang, and W. Shi. “Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network”. In: *Computer Vision and Pattern Recognition*. 2017.

[44] Gottfried W. Leibniz. “Memoir using the chain rule (Cited in TMME 7:2&3 p 321-332, 2010)”. In: (1676).

[45] Guillaume F. A. L'Hôpital. *Analyse des infiniment petits, pour l'intelligence des lignes courbes*. Paris: L'Imprimerie Royale, 1696.

[46] Jiwei Li, Will Monroe, Tianlin Shi, Alan Ritter, and Dan Jurafsky. “Adversarial Learning for Neural Dialogue Generation”. In: *arXiv preprint arXiv:1701.06547* (2017).

[47] Xiaodan Liang, Zhiting Hu, Hao Zhang, Chuang Gan, and Eric P Xing. “Recurrent Topic-Transition GAN for Visual Paragraph Generation”. In: *arXiv preprint arXiv:1703.07022* (2017).

[48] Chris J Maddison, Andriy Mnih, and Yee Whye Teh. “The Concrete Distribution: A Continuous Relaxation of Discrete Random Variables”. In: *arXiv preprint arXiv:1611.00712* (2016).

[49] Xudong Mao, Qing Li, Haoran Xie, Raymond YK Lau, and Zhen Wang. “Least Squares Generative Adversarial Networks”. In: *arXiv preprint arXiv:1611.04076* (2016).

[50] Bart van Merriënboer, Dzmitry Bahdanau, Vincent Dumoulin, Dmitriy Serdyuk, David Warde-Farley, Jan Chorowski, and Yoshua Bengio. “Blocks and Fuel: Frameworks for deep learning”. In: *arXiv preprint* abs/1506.00619 (2015). URL: http://arxiv.org/abs/1506.00619.

[51] Luke Metz, Ben Poole, David Pfau, and Jascha Sohl-Dickstein. “Unrolled Generative Adversarial Networks”. In: *arXiv preprint arXiv:1611.02163* (2016).

[52] Shakir Mohamed. *Building Machines that Imagine and Reason: Principles and Applications of Deep Generative Models (Invited Lecture at the Deep Learning Summer School)*. http://shakirm.com/slides/DLSummerSchool_Aug2016_compress.pdf. Accessed: 2017-08-15.

[53] Shakir Mohamed and Balaji Lakshminarayanan. “Learning in Implicit Generative Models”. In: *arXiv preprint arXiv:1610.03483* (2016).

[54] Hariharan Narayanan and Sanjoy K. Mitter. “Sample Complexity of Testing the Manifold Hypothesis”. In: *Advances in Neural Information Processing Systems*. 2010, pp. 1786–1794.

[55] Anh Nguyen, Jeff Clune, Yoshua Bengio, Alexey Dosovitskiy, and Jason Yosinski. “Plug & Play Generative Networks: Conditional Iterative Generation of Images in Latent Space”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. IEEE. 2017.

[56] Augustus Odena, Christopher Olah, and Jonathon Shlens. “Conditional image synthesis with auxiliary classifier gans”. In: *arXiv preprint arXiv:1610.09585* (2016).

[57] Aäron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. “Pixel Recurrent Neural Networks”. In: *International Conference on Machine Learning (ICML)*. 2016.

[58] Aäron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. “Wavenet: A generative model for raw audio”. In: *CoRR abs/1609.03499* (2016).

[59] Aäron van den Oord, Nal Kalchbrenner, Oriol Vinyals, Lasse Espeholt, Alex Graves, and Koray Kavukcuoglu. “Conditional Image Generation with PixelCNN Decoders”. In: *arXiv preprint arXiv:1606.05328* (2016).

[60] Aáron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. “WaveNet: A generative model for raw audio”. In: *arXiv preprint arXiv:1609.03499* (2016).

[61] John William Paisley, David M. Blei, and Michael I. Jordan. “Variational Bayesian Inference with Stochastic Search.” In: *ICML*. 2012.

[62] Emanuel Parzen. “On Estimation of a Probability Density Function and Mode”. In: *Ann. Math. Statist.* 33.3 (Sept. 1962), pp. 1065–1076. DOI: 10.1214/aoms/1177704472. URL: http://dx.doi.org/10.1214/aoms/1177704472.

[63] Ben Poole, Alexander A Alemi, Jascha Sohl-Dickstein, and Anelia Angelova. “Improved generator objectives for GANs”. In: *arXiv preprint arXiv:1612.02780* (2016).

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

[65] Scott Reed, Zeynep Akata, Xinchen Yan, Lajanugen Logeswaran, Bernt Schiele, and Honglak Lee. “Generative Adversarial Text-to-Image Synthesis”. In: *Proceedings of The 33rd International Conference on Machine Learning*. 2016.

[66] Jeffrey Regier, Andrew C. Miller, Jon McAuliffe, Ryan P. Adams, Matthew D. Hoffman, Dustin Lang, David Schlegel, and Prabhat. "Celeste: Variational inference for a generative model of astronomical images". In: *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*. 2015, pp. 2095–2103. URL: http://jmlr.org/proceedings/papers/v37/regier15.html.

[67] Danilo Jimenez Rezende and Shakir Mohamed. "Variational Inference with Normalizing Flows". In: *International Conference on Machine Learning (ICML)*. 2015.

[68] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. "Stochastic Backpropagation and Approximate Inference in Deep Generative Models". In: *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014*. 2014, pp. 1278–1286.

[69] Herbert Robbins and Sutton Monro. "A Stochastic Approximation Method". In: *Ann. Math. Statist.* 22.3 (Sept. 1951), pp. 400–407.

[70] Jason Tyler Rolfe. "Discrete Variational Autoencoders". In: *arXiv preprint arXiv:1609.02200* (2016).

[71] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. "Learning representations by back-propagating errors". In: *Nature* 323 (Oct. 1986), pp. 533–536. DOI: 10.1038/323533a0.

[72] Ruslan Salakhutdinov and Iain Murray. "On the Quantitative Analysis of Deep Belief Networks". In: *Proceedings of the International Conference on Machine Learning*. Vol. 25. 2008.

[73] 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*. 2016, pp. 2234–2242.

[74] Tim Salimans, Andrej Karpathy, Xi Chen, and Diederik P Kingma. "PixelCNN++: Improving the PixelCNN with discretized logistic mixture likelihood and other modifications". In: *arXiv preprint arXiv:1701.05517* (2017).

[75] Shibani Santurkar, David Budden, and Nir Shavit. "Generative Compression". In: *arXiv preprint arXiv:1703.01467* (2017).

[76] K. Simonyan and A. Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *CoRR* abs/1409.1556 (2014).

[77] Casper Kaae Sønderby, Tapani Raiko, Lars Maaløe, Søren Kaae Sønderby, and Ole Winther. "Ladder Variational Autoencoders". In: *arXiv.org* (Feb. 2016). arXiv: 1602.02282v3 [stat.ML].

[78] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *J. Mach. Learn. Res.* 15.1 (2014), pp. 1929–1958. ISSN: 1532-4435.

[79] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. "Intriguing properties of neural networks". In: *CoRR* abs/1312.6199 (2013).

[80] David Martinus Johannes Tax. "One-class classification: Concept learning in the absence of counter-examples". PhD thesis. Technische Universiteit Delft.

[81] Theano Development Team. "Theano: A Python framework for fast computation of mathematical expressions". In: *arXiv e-prints* abs/1605.02688 (May 2016). URL: http://arxiv.org/abs/1605.02688.

[82] Lucas Theis, Aäron van den Oord, and Matthias Bethge. "A note on the evaluation of generative models". In: *International Conference on Learning Representations*. 2016. URL: http://arxiv.org/abs/1511.01844.

[83] Tijmen Tieleman and Geoff Hinton. *Lecture 6.5 - RMSProp, COURSERA: Neural Networks for Machine Learning.* 2012.

[84] Cédric Villani. *Optimal Transport: Old and New.* Grundlehren der mathematischen Wissenschaften. Springer Berlin Heidelberg, 2008. ISBN: 9783540710509. URL: https://books.google.ca/books?id=hV8o5R7_5tkC.

[85] Dilin Wang and Qiang Liu. "Learning to draw samples: With application to amortized mle for generative adversarial learning". In: *arXiv preprint arXiv:1611.01722* (2016).

[86] David Warde-Farley and Yoshua Bengio. "Improving Generative Adversarial Networks With Denoising Feature Matching". In: 2017. URL: https://openreview.net/forum?id=S1X7nhsxl.

[87] Paul J. Werbos. "Applications of Advances in Nonlinear Sensitivity Analysis". In: *Proceedings of the 10th IFIP Conference, 31.8 - 4.9, NYC.* 1981, pp. 762–770.

[88] Paul J. Werbos. "Backpropagation Through Time: What It Does and How to Do It". In: *Proceedings of the IEEE* 78.10 (1990), pp. 1550–1560.

[89] Ronald J Williams. "Simple statistical gradient-following algorithms for connectionist reinforcement learning". In: *Machine learning* 8.3-4 (1992), pp. 229–256.

[90] Yuhuai Wu, Yuri Burda, Ruslan Salakhutdinov, and Roger Grosse. "On the Quantitative Analysis of Decoder-Based Generative Models". In: *arXiv preprint arXiv:1611.04273* (2016).

[91] Zhen Yang, Wei Chen, Feng Wang, and Bo Xu. "Improving Neural Machine Translation with Conditional Sequence Generative Adversarial Nets". In: *arXiv preprint arXiv:1703.04887* (2017).

[92] Raymond A. Yeh*, Chen Chen*, Teck Yian Lim, Schwing Alexander G., Mark Hasegawa-Johnson, and Minh N. Do. "Semantic Image Inpainting with Deep Generative Models". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.* * equal contribution. 2017.

[93] Fisher Yu, Yinda Zhang, Shuran Song, Ari Seff, and Jianxiong Xiao. "LSUN: Construction of a Large-scale Image Dataset using Deep Learning with Humans in the Loop". In: *CoRR* abs/1506.03365 (2015).

[94] Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. "Seqgan: sequence generative adversarial nets with policy gradient". In: *arXiv preprint arXiv:1609.05473* (2016).

[95] Matthew D. Zeiler, Dilip Krishnan, Graham W. Taylor, and Rob Fergus. "Deconvolutional networks". In: *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR 2010.* 2010, pp. 2528–2535.