Unsupervised Learning Under Uncertainty

by

Michaël Mathieu

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

Department of Computer Science

New York University

September 2017

Professor Yann LeCun

Dedication

To my family.

Acknowledgements

I would like to thank my advisor Yann LeCun for his long-timed support. Working with him has been a positive, fruitful and edifying experience. I enjoyed every discussion with him, whether it was about technical computer hacks or metaphysics - and physics - of machine learning.

I am grateful to many people over my PhD years. Koray Kavukcuoglu, Y-Lan Boureau and Pierre Sermanet took the time to teach me deep learning when I first interned at NYU, and Marco Scoffier and Clément Farabet when I returned a year later. Mikael Henaff has been a great officemate and friend, and was at the origin of many inspiring conversations. I want to thank Ross Goroshin, Pablo Sprechmann and Joan Bruna for all the things I learned working with them. Many thanks to Sixin Zhang, Arnaud Chauveur, Aditya Ramesh, Sainbayar Sukhbaatar, Emily Denton, Kyunghyun Cho and Rob Fergus for all the interesting discussions and ideas. I thank Camille Couprie for her continuous help and friendship, and her perseverance at all times, and all the "Facebook crowd" in Paris, New York, and Menlo Park. I thank Jake Zhao for being a great coder and researcher, whose help has been invaluable many times. Thanks to Yacine Jernite who has been an amazing friend, roommate, and officemate over the years.

I want to thank my preparatory classes teachers, in particular Jean-Marie Le Gluher, the most talented teacher I had. Finally, I grateful to my family. My parents, who made me pursue a research path, and Tian for all her loving support.

Abstract

Deep learning, in particular neural networks, achieved remarkable success in the recent years. However, most of it is based on supervised learning, and relies on ever larger datasets, and immense computing power. One step towards general artificial intelligence is to build a model of the world, with enough knowledge to acquire a kind of "common sense". Representations learned by such a model could be reused in a number of other tasks. It would reduce the requirement for labeled samples and possibly acquire a deeper understanding of the problem. The vast quantities of knowledge required to build common sense preclude the use of supervised learning, and suggest to rely on unsupervised learning instead.

The concept of *uncertainty* is central to unsupervised learning. The task is usually to learn a complex, multimodal distribution. Density estimation and generative models aim at representing the whole distribution of the data, while predictive learning consists of predicting the state of the world given the context and, more often than not, the prediction is not unique. That may be because the model lacks the capacity or the computing power to make a certain prediction, or because the future depends on parameters that are not part of the observation. Finally, the world can be chaotic of truly stochastic. Representing complex, multimodal continuous distributions with deep neural networks is still an open problem.

In this thesis, we first assess the difficulties of representing probabilities in high dimensional spaces, and review the related work in this domain. We then introduce two methods to address the problem of video prediction, first using a novel form of linearizing auto-encoder and latent variables, and secondly using Generative Adversarial Networks (GANs). We show how GANs can be seen as trainable loss functions to represent uncertainty, then how they can be used to disentangle factors of variation. Finally, we explore a new non-probabilistic framework for GANs.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Deep learning has achieved remarkable success in the recent years, through deep artificial neural networks. In particular, deep learning has revolutionized tasks such as object recognition [LeCun et al., 1998, Krizhevsky et al., 2012, Sermanet et al., 2013, He et al., 2015], object localization and image segmentation [Farabet et al., 2013, Sermanet et al., 2013, Lin et al., 2014], speech recognition [Hannun et al., 2014], machine translation [Sutskever et al., 2014], games AI [Silver et al., 2016, Mnih et al., 2013], and a number of other tasks [LeCun et al., 2015]. Most of these successes are based on supervised learning, which means the learning target is typically a *label*, that has been manually annotated. Since the labels are chosen so that most humans agree on them, the problem is well defined and there is little uncertainty in what the model should produce.

Supervised learning, however, relies on ever larger datasets, and immense computing power. Labeling millions of samples comes at great cost, and solving each new task independently is very inefficient. On the other hand, an intelligent system such as the human brain seems able to reuse the knowledge learned on previous

tasks very efficiently, so that humans can learn to recognize new objects, faces or even concepts given very few labeled samples, often only one.

For years, unsupervised pretraining has been seen as the main solution to this problem [Bengio et al., 2013], but it is still rarely deployed in large scale systems. The idea behind pretraining is that, instead of learning a new model from scratch for each new labeled dataset, we first train a model for another task, for which we have abundant data, then fine-tune the existing model using the labels. We rely on the *manifold* assumption to explain why this approach should work: although natural data is composed of high dimensional inputs (millions of dimensions for images), these dimensions are far from independent. In this space, the set of points that correspond to natural inputs is much smaller than the whole space. It is usually considered as a low-dimensional manifold, since there exist ways to continuously morph each realistic point into a neighboring realistic point. If the pretrained model is able to capture this manifold, the complexity of the problem can be drastically reduced, requiring less labeled samples.

However, a related but slightly different approach has recently been more successful than unsupervised pretraining: *supervised transfer learning*. It appears that some supervised tasks, such as image classification, can also be used to pretrain models that can then perform well in different contexts. Deep models trained on the ImageNet dataset [Deng et al., 2009, Russakovsky et al., 2015] have transferred surprisingly well to other visual tasks, such as classification on other vision datasets [Donahue et al., 2013, Razavian et al., 2014], detection [Girshick et al., 2013,Sermanet et al., 2013], action recognition [Simonyan and Zisserman, 2014,Dai et al., 2015], optical flow [Weinzaepfel et al., 2013], image captioning [Donahue et al., 2014, Karpathy and Li, 2014] and others [LeCun et al., 2015]. This is far

from a solved problem, however, and although the model seems to transfer well for *perception* tasks, the situation is different in other domains such as planning or reasoning. It is difficult to argue that the deep representation learned by a model trained on ImageNet has *common sense*. For instance, it is surprisingly easy to trick an ImageNet model into mislabeling an image with very high confidence, by using adversarial attacks [Szegedy et al., 2013]. One possible reason for this is that the representation is limited by the size of the dataset, which in turn is limited by the cost of labeling. It is also possible that the classification task is too easy to learn anything more than basic perceptive features.

A first step towards general AI is building a system that can learn a *model of the world*. Such a system can harness the knowledge it already possesses in order to learn new tasks with little supervision. However, learning such a model might require an immense amount of data. While there is no obvious limit to the computing power of hardware (which reliably follows Moore's law), the size of any labeled dataset is limited by the number of human beings who can work on labeling. Unsupervised learning, on the other hand, does not necessitate human time to build the dataset and can therefore scale with the speed of computers. Additionally, it would be difficult to create a labeled dataset of "the world", or a dataset of "common sense": it is unclear how to define labels and which value they should have. Unsupervised learning does not need manually annotated data and therefore works around this problem.

A model of the world would be able to understand general principles, such as gravity, collisions, human interactions, etc. We can evaluate the capacity of the model by performing *predictions*. If the model is able to make a correct prediction, such as the image formed on a camera sensor after 10 seconds, or the number of

people in a house given the schedule of the residents, it would prove its capacity to understand the world. Incidentally, this is the core principle of the scientific method: by observing the world, scientists build a theory, which in turn is used to make predictions. Confronting the predictions to the reality can either refute or reinforce the theory. In order to learn a model of the world, we can directly train our system to make predictions, as an objective. It is key to notice that predictions are by nature *probabilistic*, or at least *uncertain*. One can rarely be absolutely certain of the future state of the world. There may exist intrinsically probabilistic futures, due to quantum phenomena, for instance. Even if the world were perfectly deterministic given its global state, it would still be impossible to record this global state perfectly. Not only every sensor has noise, but it only measures a small fraction of the world. It would then be very likely that the future depends on unobserved quantities (such as the butterfly flapping its wings in the butterfly effect). Finally, even with perfect observations, the dependencies may be so complicated that the model simply does not have the "reasoning" capacity to reach certainty. In this case, probabilistic predictions would be approximating the deterministic outcome.

More generally, machine learning models represent uncertain, probabilistic outcomes. In supervised machine learning, labels are manually annotated, so the target distribution is usually cleaner, often unimodal. A unimodal distribution may be approximated by a normal distribution, which can be represented by its mean. This is why deterministic predictions are most of the time valid in regression, for instance. Complicated unsupervised tasks require the model to have the capacity to represent a more general multimodal probability distribution.

The issue of representing probabilities is a core problem of deep learning. Train-

ing a neural network typically involves minimizing a loss function. The most successful loss functions are usually built using a simple process: assuming the output of the network parametrizes a distribution, and maximizing the likelihood of the training data under this distribution. In the case of classification, the categories are discrete. Therefore, the distribution can be exactly parametrized as a categorical distribution (also known as multinoulli), using a finite number of dimensions. In the case of regression, however, the space of possible distributions is of infinite dimension and therefore the choice of the loss matters. The Mean Square Error (MSE) is the most common loss in this case. It corresponds to parametrizing a normal distribution of unit variance by its mean. Approximating a target density with a single Gaussian is only reasonable for a unimodal distribution, and will notably fail for a distribution with more than one mode.

1.1 Representing continuous probabilities

Machine learning is the process of discovering a probability distribution from the data. In this section, we define general notations that will apply to a wide class of problems, from supervised to unsupervised learning. Let Ω be a vector space containing data points. We assume that the data are drawn from an underlying probability distribution p_{data} over the space Ω. For simplicity, we identify the probability law by its associated distribution (a generalized function). We split the coordinates of the data points, so every single data point is composed of two sets of coordinates x and y, such that $(x, y) \in \Omega$. Let $\mathcal{D}$ be the training set, i.e. a finite collection of points $\{(x_1, y_1), \ldots, (x_N, y_N)\}$ independently drawn from p_{data}. We use capital letters for random variables and lowercase for actual points. We define

a learning task as estimating the parameters θ of a model, such that the model learns to represent an approximation of the conditional distribution $p_{\text{data}}(Y|X)$. We denote $\hat{p}_\theta$ the distribution represented by the model, thus learning is about finding the optimal θ^* such that $\hat{p}_{\theta^*}(Y|X) \approx p_{\text{data}}(Y|X)$. With this definition, we group density estimation (where X is empty), predictive learning (where Y is the part to predict from the context X) and supervised learning (where Y is the label). The distinction between supervised and predictive learning is blurry and we call the task of predicting a manually annotated label supervised learning.

If Y is not a discrete variable, using a neural network to learn the conditional distribution $p_{\text{data}}(Y|X)$ requires a fundamental choice. Indeed, the space of continuous distribution is of infinite dimension, so there is no finite parametrization covering the whole space of continuous distributions. There are at least three different approaches to this problem, and some of their most common instantiations will be described in Chapter 2:

1. Explicit parametrization A fixed set $\mathcal{H}$ of probability distributions is chosen, parametrized by a vector of finite dimension. For instance, $\mathcal{H}$ can be the set of normal distributions with unit variance. The model is trained to learn a function $f_\theta(x)$ that parametrizes $\mathcal{H}$. The choice of $\mathcal{H}$ is very important, and will determine the amount of bias added to the loss. This bias corresponds to the error made by approximating $p_{\text{data}}(Y|X)$ by the best candidate in $\mathcal{H}$. One advantage of this method is that single evaluation of the model returns a full conditional distribution. The space $\mathcal{H}$ is usually chosen so that the distributions are simple enough and have analytical properties, to facilitate tasks such as maximum a posteriori (MAP) estimation, or sampling from $\hat{p}_\theta(Y|X = x)$, for instance. The main disadvantage

is the introduced bias, that can be significant if a wrong class of distributions is used.

2. Implicit function Energy-based models [Lecun et al., 2006] directly train a neural network to learn a function $f_\theta(x, y)$ that corresponds to $p_{\text{data}}(Y = y|X = x)$ (which is a single scalar). While it is usually intractable to learn the density $p_{\text{data}}(Y = y|X = x)$, because the normalization constant is unknown, energy functions (unnormalized densities) can be learned. Assuming the network has infinite capacity, this method can represent any probability distribution. However, training such a network can be difficult. Although it is difficult to access the actual density $p_{\text{data}}(Y|X)$ from an energy function, we can sample from it using Monte Carlo methods, and particularly MCMC to draw samples from unnormalized distributions.

3. Generative models Generative models learn to directly produce samples from $p_{\text{data}}(Y|X = x)$ for a given x. They have a source of randomness from a known distribution (usually an input of the model), which is mapped to an approximation of the data distribution. Again, if the network has enough capacity, there is no intrinsic bias to the represented distribution. The difference with the implicit function approach is that sampling is now trivial, but the ability to compare the likelihood of events is lost. While the likelihood under an energy models is difficult to evaluate, it is impossible with a purely generative model. Evaluation and comparison of generative models are still open problems.

1.2 Learning manifold structure and factors of variation

Regardless of which method we use to model $p_{\text{data}}(Y|X)$, the task may, at first glance, seem hopeless. Indeed, the dimensionality of X and/or Y is usually very high. It is common to have images with millions of pixels, or audio sampled with tens of thousand of points per second. The well-known *curse of dimensionality* states that the number of states grows exponentially with the number of dimensions. Even with large training datasets, the fraction of the space covered by the datapoints is insignificant. Moreover, the "No Free Lunch" [Wolpert, 1996] theorem states that, in the absence of assumptions on p_{data}, we cannot hope to achieve any significant result.

Fortunately, the distribution p_{data} is not unconstrained: we can formulate *prior*, which is incorporate into the model. A common belief is that the "natural" data, encountered in the real world, lie on a very low dimensional manifold, compared to the dimensionality of the space [Cayton, 2005, Narayanan and Mitter, 2010]. For example, images composed of white noise (uncorrelated pixels) almost never correspond to natural images, as shown in Figure 1.1. This can be partly explained by another belief about natural data: it is built using a hierarchical process. At each level of the process, only a few degrees of freedom are allowed, resulting in a final manifold that has at most as many dimensions as the sum of the degrees of freedom. The mapping from these degrees of freedom to points of Ω is usually complex and highly non-linear. However, if one manages to identify these *factors of variation*, the dataset can then be represented in a space of much lower dimension, and much lower complexity. Let $\mathcal{M}$ denote the manifold associated with p_{data}.

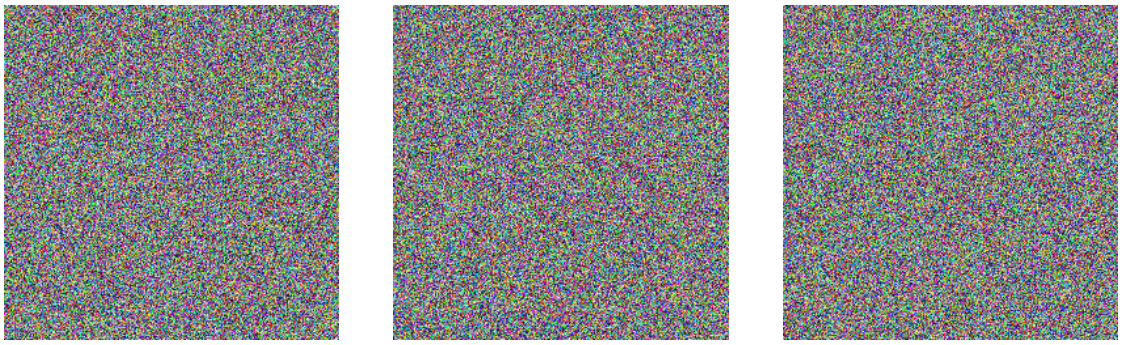


Figure 1.1: Images made of random white noise: uncorrelated pixels do not form realistic images. Points on the manifold of realistic images present highly correlated pixels.

The manifold assumption can be reformulated as stating that the vast majority of Ω has a near-zero probability under p_{data}, and only $\mathcal{M}$ contains points with significant probability.

In this perspective, machine learning can be understood as training a model to disentangle factors of variation. For instance, in the case of supervised learning, the only factor of variation we are interested in is the label. Classification is the task of identifying a specific factor of variation, the class, from the rest. Factors of variation may be considerably more diverse: in the case of video, for instance, the camera position and lighting, but also the presence or absence of different objects, are factors of variation. Disentangling factors of variations means two things. On one hand, we need to identify, explicitly or implicitly, the low-dimensional manifold $\mathcal{M}$. On the other hand, we identify and characterize the "principal" directions on $\mathcal{M}$, corresponding to factors of variation. Although it is difficult to exactly define this concept in general, ideally each "principal" direction should correspond to a degree of freedom in the process of generating the data (if such a process exists). Factors of variation can also be identified as being responsible for the continuity

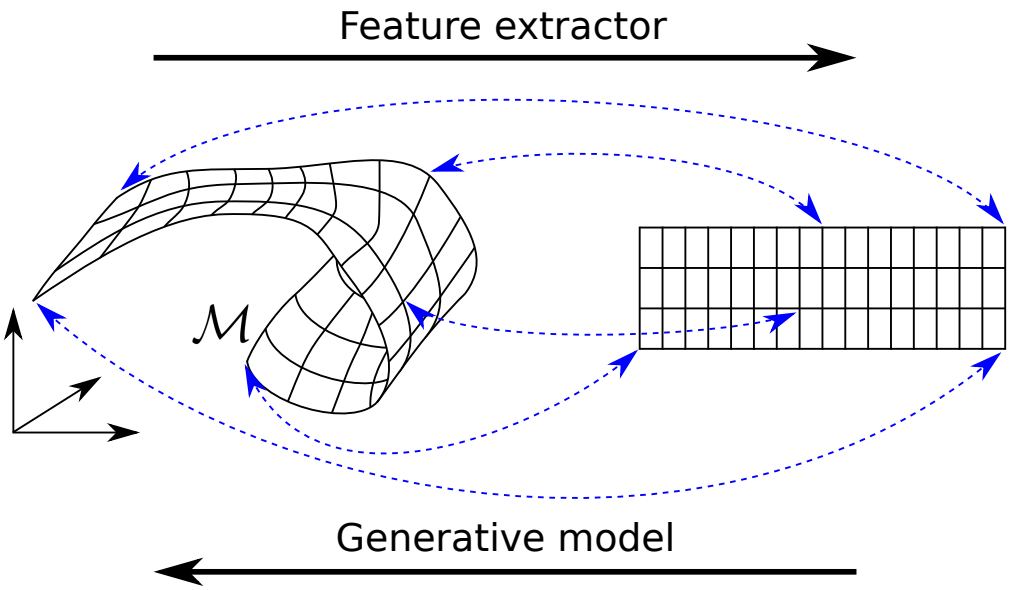


Figure 1.2: Learning the data manifold. Left: data manifold $\mathcal{M}$. Right: low-dimensional space. Blue dotted arrows: mapping learned by the model. Mapping from $\mathcal{M}$ to a low dimensional space $\mathbb{R}^n$ corresponds to extracting features of interest, while mapping from $\mathbb{R}^n$ to $\mathcal{M}$ is a generative process.

or regularity in the data: if one moves along such a direction, a single property of the data should change, preferably smoothly, while most aspects of the data stay constant. Characterizing these directions often means reparameterizing the manifold, such that we have a one-to-one mapping from the manifold to a low-dimensional vector space, and the coordinates of this space correspond to factors of variation on the manifold. This concept is illustrated in Figure 1.2.

1.3 Notations for deep neural networks

In this work, we present different neural networks, with different shapes and sizes. In order to offer consistent, easy to read notations, we define a format to describe neural networks.

A deep convolutional network is composed of successive layers. Typically, most of the layers are made of convolutions interlaced with non-linearities, and the last ones are fully connected layers. We describe a convolution with kernel $k \times k$ by the arrow $\xrightarrow[k]{\text{conv}}$ and we describe similarly non-linearities and other layers, as explained in Table 1.1. When describing a whole network, we interlace these arrows with the number of feature planes for convolutional network (where it is relevant), and the number of units for fully connected layers (which correspond to feature planes of size 1×1). When it is relevant, we also display the spatial size of the feature planes below the number of feature planes. For instance, with these notations, LeNet-5 from [LeCun et al., 1998] can be described as

$$\underset{32\times32}{1} \xrightarrow[5]{\text{conv}} \underset{28\times28}{6} \xrightarrow[2]{\text{MaxPool}} \xrightarrow{\text{Tanh}} \xrightarrow[5]{\text{conv}} \underset{10\times10}{16} \xrightarrow[2]{\text{MaxPool}} \xrightarrow{\text{Tanh}} \xrightarrow[5]{\text{conv}} \underset{1\times1}{120} \xrightarrow{\text{Tanh}} \xrightarrow{\text{FC}} 84 \xrightarrow{\text{Tanh}} \xrightarrow{\text{FC}} 10$$

and AlexNet [Krizhevsky et al., 2012] with Batch Normalization can be described with the following notation (here we do not display the spatial size of the feature planes):

$$3 \xrightarrow[11/4]{\text{convp}} \xrightarrow[\text{BN}]{\text{ReLU}} 96 \xrightarrow[5]{\text{convp}} \xrightarrow[2]{\text{MaxPool}} \xrightarrow[\text{BN}]{\text{ReLU}} 256 \xrightarrow[3]{\text{convp}} \xrightarrow[\text{BN}]{\text{ReLU}} 384 \xrightarrow[3]{\text{convp}} \xrightarrow[\text{BN}]{\text{ReLU}} 384$$
$$\xrightarrow[3]{\text{convp}} \xrightarrow[\text{BN}]{\text{ReLU}} 256 \xrightarrow[2]{\text{MaxPool}} \xrightarrow[\text{BN}]{\text{ReLU}} \xrightarrow{\text{FC}} 4096 \xrightarrow{\text{ReLU}} \xrightarrow{\text{FC}} 4096 \xrightarrow{\text{ReLU}} \xrightarrow{\text{FC}} 1000 \ .$$

1.4 Thesis Outline and Summary of Contributions

The thesis is organized as follows. In chapter 2, we develop the problem of uncertainty in machine learning and review the prior work on this aspect, along with related work on video prediction and disentanglement of factors of variation.

$\xrightarrow[k]{\text{conv}}$	Valid convolution with a $k \times k$ kernel
$\xrightarrow[k]{\text{convp}}$	Padded (full) convolution with a $k \times k$ kernel
$\xrightarrow[k/s]{\text{convp}}$	Padded (full) convolution with a $k \times k$ kernel, and stride s
$\xrightarrow{\text{ReLU}}$	Rectified Linear Unit layer
$\xrightarrow[\text{BN}]{\text{ReLU}}$	Rectified Linear Unit after Batch Normalization
$\xrightarrow[\text{BN}]{\text{LeakyReLU}}$	Leaky Rectified Linear Unit after Batch Normalization
$\xrightarrow{\text{Tanh}}$	Hyperbolic tangent layer
$\xrightarrow{\sigma}$	Sigmoid layer
$\xrightarrow[p]{\text{MaxPool}}$	Max-pooling layer with pooling windows of size $p \times p$
$\xrightarrow[p]{\text{UpSample}}$	Upsampling layer of factor $p \times p$
$\xrightarrow{\text{FC}}$	Fully connected layer
$\xrightarrow{\text{LookUp}}$	Look-up table layer
$\xrightarrow{\text{concat}}$	Concatenation of the feature planes
$\xrightarrow{\text{reshape}}$	Reshaping of the feature planes
$\xrightarrow[k/s]{\text{PhasePool}}$	Phase pooling layer (see Section 3.2.1) of size k and stride s
$\xrightarrow{\text{UnPool}}$	Unpooling layer (see Section 3.2.1)

Table 1.1: Notations used to describe different layers of deep convolutional networks.

Chapter 3 introduces the goal of video prediction as a special form of regularizer for auto-encoders, along with a way to handle uncertainty as latent variables and a "phase-pooling" operator. In Chapter 4 continues exploring video prediction, this time using an adversarial loss. Chapters 5 and 6 focus more closely on Generative Adversarial Networks. In chapter 5, we use adversarial networks along with variational auto-encoders to disentangle factors of variation without explicit data alignment, and in chapter 6, we introduce a new energy-based framework for Generative Adversarial Networks. Finally, Chapter 7 concludes the thesis and points future research directions.

Chapter 2

Motivations and Related Work

This chapter presents previous works related to this thesis. In section 2.1, we detail different unsupervised learning methods in the context of representing multi-modal distributions with a neural network. In section 2.2, we present related work for predictive models and particularly for video prediction. Finally, section 2.3 describes works in the context of disentangling factors of variation.

2.1 Representing uncertainty

Supervised learning deals with learning a distribution of the label Y given an input x. This label is determined manually, by a "supervisor", and the labeling process is fairly deterministic. Consequently, supervised tasks are typically easier than unsupervised problems. They often exhibit a distribution $p_{\text{data}}(Y|X = x)$ that has a single mode. When the labels are continuous, their distribution can be safely assumed as unimodal, so the MSE loss can be used. If the values that y can take are discrete, their probabilities are often represented using a categorical distribution.

Unsupervised learning, on the other hand, aims at discovering a hidden structure in the data. This means modeling complex and multimodal distributions. Thus, unsupervised learning relies on the ability of the model to represent distributions, particularly multimodal distributions. This intuition can be factored into the concept of *uncertainty*. In the context of predictive models, the different possible outcomes correspond to uncertainty in the prediction. Representing uncertainty is critical for the network to perform tasks correctly. The next step would be to *explain* the cause of uncertainty, as a latent factor of variation in the data. This requires an even deeper understanding of the world by the model.

In this section, we detail the different approaches to uncertainty that were presented in section 1.1 and present several existing implementations of these approaches.

2.1.1 Direct distribution parametrization

Directly parametrizing a distribution means that the output of the network, a finite dimensional vector, is used to parametrize a distribution within a pre-specified class of probability distributions. The majority of machine learning algorithms use this technique, implicitly or explicitly. For instance, discrete cross-entropy or mean square error losses fall into this framework. In most supervised cases, it is assumed that the distribution is simple and unimodal, but this is not necessary and in some cases, the approach can be extended to model complex distributions.

Probability distributions over discrete variable (with finite number of values) can be fully – but not necessarily efficiently – represented with a categorical distribution. We only need to specify the probability of each value to characterize the whole distribution. Therefore, when Y is discrete, the network usually produces

a distribution through a softmax operation. Minimizing the cross-entropy loss is equivalent to maximizing the likelihood of the data under the learned distribution.

On the other hand, the distribution $p_{\text{data}}(Y|X = x)$ cannot be parametrized exactly when Y is continuous. A natural idea is to approximate the true distribution $p_{\text{data}}(Y|X = x)$ by a distribution in a class of simpler, tractable distributions, parametrized by a finite vector $\phi_x \in \mathbb{R}^p$. We denote $q_{\phi_x}(Y)$ this distribution, and $\mathcal{H}$ the set of representable distributions, *i.e.* $\mathcal{H} = \{q_\phi | \phi \in \mathbb{R}^p\}$. The task is to generate a vector ϕ_x for each conditioning variable x, by learning parameters θ so that for all x, the vector $\phi_x = f_\theta(x)$ parametrizes an approximation of $p_{\text{data}}(Y|X = x)$. Since we only have access to individual samples of $p_{\text{data}}(Y|X = x)$, it is natural to use a maximum likelihood approach. The desired weight vector θ^* is defined by maximizing the log-likelihood:

$$\theta^* = \arg\max_\theta \sum_{(x,y)\in\mathcal{D}} \log(q_{f_\theta(x)}(y)). \tag{2.1}$$

In other words, the loss of the network, for a given input x is

$$\mathcal{L}_\theta(x) = \mathbb{E}_{y\sim p_{\text{data}}(Y|X=x)} \left[-\log(q_{f_\theta(x)}(y)) \right]. \tag{2.2}$$

For instance, if ϕ represents the mean of a Gaussian with an identity covariance matrix, then

$$\theta^* = \arg\min_\theta \sum_{(x,y)\in\mathcal{D}} ||y - f_\theta(x)||_2^2. \tag{2.3}$$

This is the Mean-Square Error (MSE). Therefore, using the MSE loss corresponds to approximating $p_{\text{data}}(Y|X = x)$ with a normal distribution with unit variance.

If we assume that ϕ corresponds to the mean of a Laplace distribution with

variance 1, we get

$$\theta^* = \arg\min_\theta \sum_{(x,y)\in\mathcal{D}} |y - f_\theta(x)|. \tag{2.4}$$

which corresponds to a L^1 loss (sometimes called Absolute Value Loss).

Parametrizing a Gaussian or a Laplace distribution is convenient: the form of the loss is simple, and it is convex with respect to $f_\theta(x)$. If $p_{\text{data}}(Y|X = x)$ is not unimodal, however, modeling it with a Gaussian or a Laplace distribution can only perform poorly.

Mixtures of Gaussians are a natural candidate to model multimodal distributions. A mixture of k Gaussians with unit-variance, parametrized by $\phi = (\alpha, \mu)$, is defined as

$$q_{(\alpha,\mu)}(x) = \sum_{i=1}^{k} \alpha_i \mathcal{G}(x - \mu_i) \tag{2.5}$$

where $\mathcal{G}$ is a zero-mean Gaussian with unit-variance, $\forall i, \alpha_i \geq 0$, and $\sum_{i=1}^{n} \alpha_i = 1$. Using such a distribution for q leads this loss:

$$\theta^* = \arg\min_\theta \sum_{(x,y)\in\mathcal{D}} \log\left(\sum_{i=1}^{k} \alpha_i(x)\mathcal{G}(y - \mu_i(x))\right) \tag{2.6}$$

$$\tag{2.7}$$

Models with such a loss are often called Mixture Density Networks [Bishop, 1994] and have been successfully applied to tasks such as speech synthesis [Zen and Senior, 2014]. They can be extended to non-unit covariance Gaussians, but the number of parameters increases quadratically with the size of the space. Diagonal covariance matrices are often seen as a good compromise as the number of parameters only grows linearly.

Instead of directly parametrizing the distribution q_ϕ, it is also possible to use

the chain rule for probabilities to decompose it, arbitrarily or not, into conditional probabilities [Uria et al., 2016]. Assuming $Y = (Y_1, Y_2, \ldots, Y_N)$, we can write

$$q_\phi(Y) = q^{(1)}_{\phi_1}(Y_1) q^{(2)}_{\phi_2}(Y_2|Y_1) \ldots q^{(N)}_{\phi_N}(Y_N|Y_1, \ldots, Y_{N-1}) \tag{2.8}$$

where $q^{(1)}, q^{(2)}, ...q^{(N)}$ are N different, or identical, distribution parameterizations. The advantage of this method is to factor the representation. Often, the model used to obtain ϕ_i is shared for all i, in the form of a Recurrent Neural Network. This scheme is widely used when Y is a discrete temporal sequence, such as text [Mikolov, 2010]. When Y takes continuous values, it is still possible to discretize it in order to use the well-behaved categorical distribution. Discretizing the values yielded good results for image generation and video prediction, by quantizing patches [Ranzato et al., 2014] or pixels, in the PixelRNN [van den Oord et al., 2016a] and PixelCNN [van den Oord et al., 2016b] models. The main drawback of this approach is computational time: the model needs to be evaluated many times in order to compute each conditional probabilities. Another issue is that in order to obtain the factored posterior distribution, it is necessary to either integrate over all the possible values of the conditioning variables, which does not scale up to high dimension, or to iteratively sample the variables from y_1 to y_N, which is not differentiable and introduces an approximation of the evaluation.

2.1.2 Implicit function and energy models

Instead of parametrizing the whole distribution $p_{\text{data}}(Y|X = x)$, we can imagine learning a function $\hat{p}_\theta(y|x)$ to approximate the distribution $p_{\text{data}}(Y = y|X = x)$. However, normalization is difficult. It is often even impossible. Indeed,

$p_{\text{data}}(Y|X = x)$ is a distribution (i.e. a generalized function), but may not be a density. In particular, if the points of non-zero probability are concentrated on a low-dimensional manifold, $p_{\text{data}}(Y|X = x)$ is not a density (it is zero almost everywhere, and infinite on the manifold). As mentioned in section 1.2, it is believed that this situation is frequent.

Energy-based models [Lecun et al., 2006] are an answer to the normalization problem. An energy can be seen as an unnormalized parametrization of the distribution. Energy-based models associate an energy value $f_\theta(x, y)$ to each pair (x, y). It is a scalar value, usually unbounded, and large energies correspond to low probabilities. Energies can be converted to probability densities by using the Gibbs distribution:

$$p_\theta(y|x) = \frac{e^{-f_\theta(x,y)}}{\int_{y'} e^{-f(x,y')\mathrm{d}y'}}. \tag{2.9}$$

The only constraint for energies to correspond to probabilities is that the denominator of Equation (2.9) is finite. Although this constraint also concerns all points of the space, it is actually easier to (weakly) enforce than the normalization constraint of a probability density, and numerous methods have been developed to that end. In case $p_{\text{data}}(Y|X = x)$ is not a density (i.e. is only non-zero on a set of measure zero), Equation (2.9) is not valid, but it can be adjusted by restricting the domain of y to $\{y|f(x, y) \neq 0\}$.

There is no groundtruth target for $f_\theta(x, y)$. The only available data are the training points, which are samples from $p_{\text{data}}(X, Y)$. The general idea to train energy models is to decrease the output energy on sample points from the training set, and increase it elsewhere. In the vocabulary of energy models, an input point for which the energy should be decreased is called a *positive sample* and a point where the energy should be increased is called a *negative sample*. For in-

stance, maximizing the log-likelihood of the data under Equation (2.9) results in the following loss:

$$\theta^* = \arg\min_\theta \mathbb{E}_{x\sim p_{\text{data}}(X)}\Big[\underbrace{\mathbb{E}_{y\sim p_{\text{data}}(Y|X=x)}[f_\theta(x,y)]}_{\text{positive samples}} - \underbrace{\mathbb{E}_{y\sim p_\theta(Y|X=x)}[f_\theta(x,y)]}_{\text{negative samples}}\Big]. \quad (2.10)$$

The positive samples are easy to obtain since they come from the training set. However, in the absence of negative samples or other constraints, positive samples are useless. Training the model from these points alone results into a collapsed state where the model attributes low energy to every point. This collapse can be avoided using different strategies, and the use of negative samples is not always necessary. If the area of low energy can be constrained, no negative samples are required: pushing the energy down on the data points automatically pushes the energy up on every other point. Unfortunately, constraining this energy to a fixed value is as hard as normalizing the probability density, and most of the models managing this have a simple form (such as PCA [Pearson, 1901], k-means [Lloyd, 1982] and ICA [Hyvärinen et al., 2004]).

It is also possible to regularize the model to limit the area of low-energy points, without explicitly knowing this area. This is, for instance, the case with sparse coding [Olshausen and Field, 1997], sparse auto-encoders [Ranzato et al., 2007b, Poultney et al., 2006, Ranzato et al., 2007c], Predictive Sparse Decomposition (PSD) [Kavukcuoglu et al., 2010a] and more generally most regularized auto-encoders [Zemel, 1994]. These models have a regularization term that makes the model produce a high energy in the absence of other constraints. Positive samples are pushing against this regularizer, resulting on low-energy on the positive samples and high energy everywhere else. We detail a few specific auto-encoder

architectures in section 2.1.2.1.

Using carefully selected negative samples to increase the energy is an alternative method. Equation (2.10) shows that in order to maximize the likelihood of the data, the negative samples should be chosen as following the distribution represented by the energy model p_θ. However, the model can only be evaluated to return the energy of a specific point, not sampling. Therefore, Monte Carlo methods, such as MCMC, can be used to produce negative samples. While MCMC methods are efficient when the distribution is unimodal, they tend to become slow and inaccurate when the modes are separated. However, this method has remained popular to obtain negative samples and is the basis of contrastive divergence [Carreira-Perpinan and Hinton, 2005]. Other approaches are possible to obtain negative samples. Energy-based adversarial networks [Kim and Bengio, 2016, Zhao et al., 2016b] use another network to produce negative samples. Denoising auto-encoders [Vincent et al., 2008] can be understood as using a noisy version of the training samples as negative samples.

2.1.2.1 Auto-encoders

Most auto-encoders can be cast into the energy-based setting, by defining the energy as the reconstruction loss. An auto-encoder is a model that is trained to learn two functions, Enc_θ and Dec_θ, such that $\text{Dec}_\theta \circ \text{Enc}_\theta$ is near the identity function on the training points. To stay coherent with the previous notations, we define conditional auto-encoders, although the conditioning variable x is often unused:

$$\forall (x, y) \in \mathcal{D}, \qquad \text{Dec}_\theta(\text{Enc}_\theta(x, y)) \approx y. \tag{2.11}$$

The energy is then defined by:

$$f_\theta(x, y) = ||\text{Dec}_\theta(\text{Enc}_\theta(x, y)) - y||_2^2. \tag{2.12}$$

Auto-encoders must learn the identity function on the training set $\mathcal{D}$, and ideally on the data manifold $\mathcal{M}$, but not over the whole space. If they did, the learned representation would most likely be useless. Moreover, they would not be an energy model, since the energy would be constantly zero. From the energy-based model point of view, the mechanism that is used to prevent the auto-encoder to learn the identity function corresponds to the mechanism that restricts the area of low energy.

Sparse auto-encoder [Ranzato et al., 2007b,Poultney et al., 2006,Ranzato et al., 2007c] and PSD [Kavukcuoglu et al., 2010a] use a sparse penalty to limit the area of low energy. The models are trained so that the code, i.e. the output of the encoder, is *sparse*. A code is said to be sparse if most of its coordinates are set to zero. The sparsity is usually obtained through a L^1 regularization over the code, but other methods exist, for instance k-sparse auto-encoders [Makhzani and Frey, 2013] threshold the top k components, and Winner-Takes-All auto-encoders [Makhzani and Frey, 2014] is the extreme version of this where only one component is activated at a time. Predictive Sparse Decomposition [Kavukcuoglu et al., 2010a] is similar to a sparse auto-encoder, but the training procedure differs: the encoder is trained to match an exact, slower, inference algorithm for the code.

Other penalties are possible in order to restrict the low-energy area. Denoising auto-encoders [Vincent et al., 2008] are explicitly trained not to be the identity function in the neighborhood of $\mathcal{M}$. Contractive auto-encoders [Rifai et al., 2011]

use a penalty on the Frobenius norm of the Jacobian of the code to achieve the same effect. Slow auto-encoders [Goroshin et al., 2015a] use a slowness penalty that forces parts of the code to stay constant on temporal neighborhoods.

The strength restricting penalty of auto-encoders is modulated through a hyper-parameter, which is adjusted on a validation set. It may be difficult to adjust in a general setting, and training deep auto-encoders requires some care. For instance, deep sparse auto-encoders work best when training a single layer of encoder and decoder at a time [Arnold and Ollivier, 2012].

2.1.3 Generative models

Learning the density $p_{\text{data}}(Y|X = x)$ may be difficult for complex, high-dimensional distributions. Moreover, it does not provide a direct mechanism to generate samples from the model (typically, sampling is done with MCMC methods, which are either slow or biased with multimodal distributions). In some cases, it may be easier, or more useful, to directly learn a sampling function. Instead of – or in addition to – learning $p_{\text{data}}(Y|X = x)$, we learn a function $f_\theta(x, \epsilon)$ such that $f_\theta(x, \epsilon) \sim p_{\text{data}}(Y|X = x)$ when $\epsilon \sim p_\epsilon(\epsilon)$, where p_ϵ is a fixed distribution (typically normal or uniform). Although there are numerous possible ways to do this, two approaches are currently popular in the deep learning community: Variational Auto-Encoders [Kingma and Welling, 2013] and Generative Adversarial Networks [Goodfellow et al., 2014]. In this section, we explain the fundamental notions of these two approaches since they will be useful in the next chapters.

2.1.3.1 Variational Auto-Encoders

Variational Auto-Encoders [Kingma and Welling, 2013] are an extension of auto-encoders. As with most auto-encoders, they can be seen as an energy model, via the reconstruction error, but they also have the ability to generate samples. The difference with regular auto-encoders is that the loss results from minimizing the *variational bound*. The model is trained so that the code z, which corresponds to the activations of a hidden layer, follows a predefined distribution (in practice, a normal distribution, thanks to the reparametrization trick).

Let us define a decoder Dec_θ parametrized by θ. We wish to model the distribution $p_{\text{data}}(Y|X = x)$ for a given x, but in order to simplify the notations, we assume we are in the pure unsupervised setting, modeling $p_{\text{data}}(Y)$, although conditioning on $X = x$ is not an issue. The end goal is that the marginal distribution $p(Y) = \int_z p_{\text{Dec}_\theta}(Y|Z = z)p_Z(Z = z)\text{d}z$ models the true distribution $p_{\text{data}}(Y)$, where $p_{\text{Dec}_\theta}(Y|Z = z)$ is given by the decoder, and the code z follows a prior distribution p_Z. So the decoder is explicitly parametrizing a distribution, which is difficult if the distribution is multimodal. However, the distribution is conditioned on Z, and while we may not be able to assume that $p_{\text{data}}(Y)$ is unimodal, we can safely assume that $p_{\text{Dec}_\theta}(Y|Z)$ is. For instance, the decoder can generate the mean and the diagonal variance of a Gaussian distribution, which corresponds to a weighted MSE loss.

Jointly learning Dec_θ and inferring Z is intractable for general models. Besides, even if Dec_θ was already trained, inferring Z given Y at test time may be difficult and slow. To that end, an encoder Enc_θ is added to the system, in order to produce the distribution of Z given a Y. Note that we parametrize it using θ to simplify the notation but it does not need to share parameters with Dec_θ, they can each

use a different set of coordinates of θ. The encoder learns an approximation of $p_{\mathrm{Dec}_\theta}(Z|Y)$. The main role of the encoder is to help training Dec_θ, and can be discarded after the training is finished if all we need is a generative model.

As usual, we compute the log-likelihood,[1] we drop the θ for clarity:

$$\begin{aligned}
\log(p_{\mathrm{Dec}}(Y)) &= \int_Z p_{\mathrm{Enc}}(Z|Y) \log(p_{\mathrm{Dec}}(Y)) \mathrm{d}Z && (2.13)\\
&= \int_Z p_{\mathrm{Enc}}(Z|Y) \log\left(\frac{p_{\mathrm{Dec}}(Y,Z)}{p_{\mathrm{Dec}}(Z|Y)}\right) \mathrm{d}Z && (2.14)\\
&= \int_Z p_{\mathrm{Enc}}(Z|Y) \left(\log\left(\frac{p_{\mathrm{Enc}}(Z|Y)}{p_{\mathrm{Dec}}(Z|Y)}\right) - \log\left(\frac{p_{\mathrm{Enc}}(Z|Y)}{p_{\mathrm{Dec}}(Y,Z)}\right)\right) \mathrm{d}Z && (2.15)\\
&= D_{\mathrm{KL}}(p_{\mathrm{Enc}}(Z|Y)||p_{\mathrm{Dec}}(Z|Y)) + \mathcal{L}(\mathrm{Enc},\mathrm{Dec},Y) && (2.16)
\end{aligned}$$

where

$$\mathcal{L}(\mathrm{Enc},\mathrm{Dec},Y) = \mathbb{E}_{Z\sim p_{\mathrm{Enc}}(Z|Y)}\left[\log(p_{\mathrm{Dec}}(Y,Z)) - \log(p_{\mathrm{Enc}}(Z|Y))\right]. \quad (2.17)$$

Unfortunately, estimating the Kullback-Leibler divergence $D_{\mathrm{KL}}(p_{\mathrm{Enc}}(Z|Y)||p_{\mathrm{Dec}}(Z|Y))$ is difficult, because $p_{\mathrm{Dec}}(Z|Y)$ cannot be evaluated. Fortunately, this quantity is always non-negative, thus we obtain

$$\log(p_{\mathrm{Dec}}(Y)) \geq \mathcal{L}(\mathrm{Enc},\mathrm{Dec},Y). \quad (2.18)$$

We see that the closer $p_{\mathrm{Enc}_\theta}(Z|Y)$ is to approximating $p_{\mathrm{Dec}_\theta}(Z|Y)$, the tighter this bound is. It makes sense to use $\mathcal{L}(\mathrm{Enc}_\theta,\mathrm{Dec}_\theta,Y)$ as a loss, since it is a lower bound

[1] This derivation only works assuming that $p_{\mathrm{Dec}_\theta}(Y,Z) \neq 0$ almost everywhere, which is true if the decoder parametrizes a Gaussian.

on the log-likelihood, called the variational bound. We can re-write it as

$$\mathcal{L}(\mathrm{Enc}_\theta, \mathrm{Dec}_\theta, Y) = \underbrace{\mathbb{E}_{p_{\mathrm{Enc}_\theta}(Z|Y)}\left[\log(p_{\mathrm{Dec}_\theta}(Y,Z))\right]}_{\text{reconstruction}} - \underbrace{D_{\mathrm{KL}}(p_{\mathrm{Enc}_\theta}(Z|Y)||p_Z(Z))}_{\text{regularizer}} \tag{2.19}$$

The two term of Equation (2.19) have a natural interpretation. The first term is a reconstruction loss, where the code Z is stochastic (hence the expected value). The second term can be seen as a regularizer, where the distribution $p_{\mathrm{Enc}_\theta}(Z|Y)$ is regularized towards $p_Z(Z)$. It seems that p_Z is unknown, but the "reparametrization trick" is used to work around this problem. Under mild assumptions, it can be assumed that $Z \sim \mathcal{N}(0, I)$ [Kingma and Welling, 2013]. The intuition behind this is that a regular distribution can be seen as a deterministic function of another distribution (in this case, a Gaussian). This is called a *push-forward* distribution, and the decoder can learn the push-forward function. Therefore, the second term of Equation (2.19) simply states that the distribution of the code Z should be close to a Gaussian distribution. The expected values in the loss are approximated using Monte-Carlo estimator, i.e. by sampling Z, and the encoder Enc_θ produces two vectors μ and σ, corresponding to the mean and the diagonal covariance matrix of a normal distribution.

The strong theory behind Variational Auto-Encoders makes them easy to train and relatively hyperparameter-free. They can be used as an energy model on Y, and at the same time can be used to generate new samples, by sampling $Z \sim \mathcal{N}(0, I)$. However, recent experiments [Mescheder et al., 2017] show that the reconstructions can be blurry, which means the modes of p_{Dec_θ} are not as separated as the modes of p_{data}, and that the reconstruction can be very different from the input, which indicates that the encoder and the decoder are not reciprocal of

each other, therefore the variational bound is not tight.

2.1.3.2 Generative Adversarial Networks

Generative Adversarial Networks (GAN) [Goodfellow et al., 2014] are a different approach to generative models. The GAN setup is composed of two models, the *generator* G and the *discriminator* D (sometimes called the adversary). The two models play a game (as in the game theory), where the generator is trained to generate samples that the discriminator cannot separate from the real ones, and the discriminator is trained to tell real samples from generated ones. Instead of trying to reach a minimum, this system is said to be trained when it reaches a stable point, where none of the two models can reduce its loss when the other is fixed. Such a stable point is known in game theory as a Nash equilibrium, i.e. a point where any change of D while keeping G fixed will result in a poorer performance of D, and any change of G while keeping D fixed will result in a poorer performance of G.

The discriminator D is a classifier model. Therefore, it is natural to use binary cross-entropy loss. Through a sigmoid, the discriminator produces a number between 0 and 1, which can be interpreted as the estimated probability that the sample comes from the real dataset.

Unfortunately, to this day, there is no efficient, systematic method to find a Nash equilibrium, and only heuristics are used. The most widely used is Alternated SGD, which consists in alternatively lowering the loss of D and the loss of G.

Let $\mathbf{D} \subset \mathcal{D}$ be a minibatch. Given a fixed generator G, we train D to separate real samples Y from generated samples $G(\epsilon)$ where $\epsilon \sim \mathcal{N}(0, I)$. Therefore, the

loss of D is given by

$$\mathcal{L}_D = \frac{1}{|\mathbf{D}|} \sum_{\substack{Y \in \mathbf{D} \\ \epsilon \sim \mathcal{N}(0,1)}} \mathcal{L}_{BCE}(D(Y), 1) + \mathcal{L}_{BCE}(D(G(\epsilon)), 0) \tag{2.20}$$

where $\mathcal{L}_{BCE}$ is the Binary Cross Entropy. The left term makes the discriminator attribute high probability to real samples, while the second term makes it attribute low probability to fake samples.

Given a fixed D, the loss for training G is

$$\mathcal{L}_G = \frac{1}{|\mathbf{D}|} \sum_{\substack{X \in \mathbf{D} \\ \epsilon \sim \mathcal{N}(0,1)}} \mathcal{L}_{BCE}(D(G(\epsilon)), 1). \tag{2.21}$$

The intuition behind this loss is that the generator should be trained to produce samples on which the discriminator attributes high probability.

It can be proven [Goodfellow et al., 2014,Arjovsky and Bottou, 2017] that, given infinite capacity, if the system composed of the generator and the discriminator reaches a Nash equilibrium, then the distribution represented by G is undistinguishable from the distribution p_{data}.

Generative Adversarial Networks have been successful, among others, in generating sharp images [Goodfellow et al., 2014, Denton et al., 2015, Radford et al., 2015,Salimans et al., 2016, Im et al., 2016], video prediction [Mathieu et al., 2015], semi-supervised learning [Salimans et al., 2016], super-resolution [Ledig et al., 2016] and image segmentation [Luc et al., 2016].

2.2 Video prediction

Unsupervised learning aims at discovering structure in the data. Although image reconstruction has been one of the main way to pretrain networks for a long time [Vincent et al., 2008, Le, 2013], advances in the ability to represent complex probability distributions, in particular through Variational auto-encoders and Generative Adversarial Networks, gives us the ability to address the problem of *prediction*. While reconstruction aims at modeling the whole distribution p_{data}, through reconstructing each input, the goal of prediction is to model $p_{\text{data}}(Y|X)$ where Y and X are two distinct parts of the data, typically Y is the future and X is the past. The sole ability of a model to predict frames in the future requires building accurate, nontrivial internal representations. This is a significant departure from the problem of image reconstruction, where the main task, reconstruction, will not lead to non-trivial solutions in the absence of other constraints (such as sparsity). Therefore, it can be postulated that the better the predictions of such a system are, the better the feature representation should be. Indeed, the work of [Srivastava et al., 2015] demonstrates that learning representations by predicting the next sequence of image features improves classification results on two action recognition datasets.

We formulate prediction as the general task of modeling $p_{\text{data}}(Y|X)$, which includes, among others, inpainting and super-resolution. From here on, however, we focus on the task of *video prediction*. We consider a dataset made of videos, where each video is composed of a sequence of frames $(x^1, x^2, \ldots, x^n)$. The task of video prediction consists of predicting the target $y = (x^m, x^{m+1}, \ldots, x^n)$ from the input $x = (x^1, x^2, \ldots, x^{m-1})$.

Top performing algorithms for action recognition exploit the temporal informa-

tion in a supervised way, such as 3D convolutional networks [Tran et al., 2015], or spatio-temporal convolutional models [Simonyan and Zisserman, 2014], which can require months of training, and heavily labeled datasets. This could be reduced using unsupervised learning. The authors in [Wang and Gupta, 2015] compete with supervised learning performance on ImageNet, by using a siamese architecture [Bromley et al., 1993] to mine positive and negative examples from patch triplets of videos in an unsupervised fashion. Unsupervised learning from video is also exploited in the work of [Vondrick et al., 2016a], where a convolutional model is trained to predict sets of future possible actions, and [Vondrick et al., 2016b] where videos are generated through a model disentangling foreground and background. [Jayaraman and Grauman, 2015] focuses on learning a feature space equivariant to ego-motion. Beside unsupervised learning, a video predictive system may find applications in robotics [Kosaka and Kak, 1992], video compression [Ascenso et al., 2005] and inpainting [Flynn et al., 2015] to name a few.

Recently, predicting future video sequences appeared in different settings: [Ranzato et al., 2014] defined a recurrent network architecture inspired from language modeling, predicting the frames in a discrete space of patch clusters. [Srivastava et al., 2015] adapted a LSTM model [Hochreiter and Schmidhuber, 1997] to future frame prediction. [Oh et al., 2015] defined an action conditional auto-encoder model to predict next frames of Atari-like games. In the two works dealing with natural images, a blur effect is observed in the predictions, due to different causes. In [Ranzato et al., 2014], the transformation back and forth between pixel and clustered spaces involves the averaging of 64 predictions of overlapping tilings of the image, in order to avoid a blockiness effect in the result. Short-term results from [Srivastava et al., 2015] are less blurry, however the L^2 loss function inher-

ently produces blurry results. [Villegas et al., 2017] use pose supervision in order to reduce uncertainty and produce sharp long term predictions with a hierarchical model based on LSTMs. By discretizing the values that each pixel can take, and predicting one pixel at a time, models such as PixelRNN [van den Oord et al., 2016a] and PixelCNN [van den Oord et al., 2016b] preserve the sharpness, at the cost of running a model for each pixel prediction, which results in slower training and testing.

2.3 Disentangling factors of variation

Learning the manifold of data and its factors of variation is another view on most machine learning algorithms. For instance, video prediction can be understood as disentangling one specific factor of variation, time, from the other ones. Given a labeled dataset, on the other hand, classification extracts one specific factor, the class, and discards the rest. It is easier to discard the other factors, but it may not be optimal in learning good representations. Indeed, the objective of a classifier is to keep the information about the classes present in the train set. If the same model is transferred on another task, however, some information that is relevant for the other task may have been discarded. Instead, learning an *equivariant* representation, with the class on one side and the other dimensions on the other side, would not discard the information but separate it instead. This idea is at the root of the concept of capsules [Hinton et al., 2011] and has been explored at length in the context of content and style disentanglement.

There is a vast literature on learning disentangled representations. Bilinear models [Tenenbaum and Freeman, 2000] were an early approach to separate con-

tent and style for images of faces and text in various fonts. What-where autoencoders [Ranzato et al., 2007c, Zhao et al., 2016a] combine discrimination and reconstruction criteria to recover the factors of variation not associated with the labels. In [Hinton et al., 2011], an auto-encoder is trained to separate a translation invariant representation from a code that is used to recover the translation information. In [Cheung et al., 2014], the authors show that standard deep architectures can discover and explicitly represent factors of variation aside those relevant for classification, by combining auto-encoders with simple regularization terms during the training. In the context of generative models, the work in [Reed et al., 2014] extends the Restricted Boltzmann Machine by partitioning its hidden state into distinct factors of variation. The work presented in [Kingma et al., 2014] uses a VAE in a semi-supervised learning setting. Their approach is able to disentangle the label information from the hidden code by providing an additional one-hot vector as input to the generative model. Similarly, [Makhzani et al., 2015] shows that auto-encoders trained in a semi-supervised manner can transfer handwritten digit styles using a decoder conditioned on a categorical variable indicating the desired digit class.

The work in [Dosovitskiy et al., 2014, Kulkarni et al., 2015] further explores the application of content and style disentanglement to computer graphics. Whereas computer graphics involve going from an abstract description of a scene to a rendering, these methods learn to go backward from the rendering to recover the abstract description. This description includes attributes such as orientation and lighting information. While these methods are capable of producing impressive results, they benefit from being able to use synthetic data, making strong supervision possible.

Closely related to the problem of disentangling factors of variations in representation learning is that of learning fair representations [Louizos et al., 2016,Edwards and Storkey, 2015]. In particular, the Fair Variational Auto-Encoder [Louizos et al., 2016] aims to learn representations that are invariant to certain nuisance factors, of variation, while retaining as much of the remaining information as possible. For instance, this method has been applied to obtain a model that predicts credit rating while being agnostic to gender. The authors propose a variant of the VAE that encourages independence between the different latent factors of variation.

The problem of disentangling factors of variation also plays an important role in completing image analogies, the goal of the end-to-end model proposed in [Reed et al., 2015]. Their method relies on having access to matching examples during training.

The next chapter explores a way to train an auto-encoder on video frames to produce codes that are aligned in the feature space. It can be seen as identifying the time component in video sequences, and disentangling it from the other factors.

Chapter 3

Learning to Linearize Videos Under Uncertainty

3.1 Introduction

In this chapter, we propose to use time as a weak form of supervision by introducing a new architecture and loss for training deep feature hierarchies that linearize the transformations observed in video sequences. We also address the problem of inherent uncertainty in prediction by introducing latent variables that are non-deterministic functions of the input into the network architecture.

Recently there has been a flurry of work on learning features from video using varying degrees of supervision [Wang and Gupta, 2015, Ranzato et al., 2014, Vondrick et al., 2016a, Srivastava et al., 2015]. If we assume that data occupies some low dimensional manifold $\mathcal{M}$ in a high dimensional space, then videos can be considered as one-dimensional trajectories on this manifold parametrized by time. Successfully training an auto-encoder on individual frames will result in the char-

acterization of the manifold of the frames. However, this will not tell us anything about the temporal structure of the frames in a given video. We can incorporate the temporal constraint by including a *linearity* constraint. If we enforce that the codes of successive frames are aligned, we can take the temporal dependencies into account while having an individual code for each frame.

We introduce a loss and architecture that addresses the problem of learning linearized frame representations, by solving a prediction objective which, unlike reconstruction, requires the model to learn a non-trivial function. The problem of uncertainty is addressed by introducing a set of latent variables that are non-deterministic functions of the input, which are used to explain the unpredictable aspects of natural videos. Finally, a new operator, called *phase-pooling* is introduced to facilitate linearization by inducing a topology on the feature space.

This work is closely related to fully unsupervised approaches for learning representations from video such as [Goroshin et al., 2015a, Kayser et al., 2001, Cadieu and Olshausen, 2012, Wiskott and Sejnowski, 2002, Mobahi et al., 2009]. It is most related to [Ranzato et al., 2014] which also trains a decoder to explicitly predict video frames. Our proposed architecture was inspired by those presented in [Ranzato et al., 2007c] and [Zeiler et al., 2010]. The phase-pooling architecture is inspired by capsules [Hinton et al., 2011], and forms the foundation of Stacked What-Where Auto-Encoders [Zhao et al., 2016a].

3.1.1 Linearizing using a prediction objective

Let Enc_θ be the encoder, a network that takes a frame x and produces a code z, and Dec_θ be the decoder, taking a code z and producing a sample $\tilde{x}$. The vector θ represent the full set of parameters of both the encoder and the decoder. Auto-

encoders are trained to satisfy the reconstruction constraint: $\text{Dec}_\theta(\text{Enc}_\theta(x)) \approx x$ when x is a real frame.

We define the *linearity* constraining, the additional constraint that codes of successive frames in a video are aligned. Let us consider sequences of three frames. The reconstruction loss, for all three frames individually, is defined as

$$\mathcal{L}^{\text{rec}}_\theta(x) = \sum_{i=1}^{3} ||\text{Dec}_\theta(\text{Enc}_\theta(x^i)) - x^i||_2^2. \tag{3.1}$$

We can naively enforce the linearity constraint by adding a second term to the loss:

$$\mathcal{L}^{\text{lin}}_\theta(x) = ||2\text{Enc}_\theta(x^2) - \text{Enc}_\theta(x^1) - \text{Enc}_\theta(x^3)||_2^2. \tag{3.2}$$

We assumed that the video is composed of three frames for simplicity, but the definitions can easily be extended to longer videos.

This would resemble Slow Feature Analysis [Goroshin et al., 2015a]. However, this approach suffers from a standard problem with auto-encoders: the pair encoder/decoder can adjust to decrease $\mathcal{L}^{\text{lin}}$ indefinitely, keeping $\mathcal{L}^{\text{rec}}$ constant and without changing the function they compute in a meaningful manner. For instance, if we replace Enc_θ by $\frac{1}{\gamma}\text{Enc}_\theta$ and Dec_θ by γDec_θ (for $\gamma > 1$), the reconstruction loss $\mathcal{L}^{\text{rec}}$ is unchanged, while the linearity loss $\mathcal{L}^{\text{lin}}$ is decreased by $\frac{1}{\gamma}$. This problem can be mitigated in the case of a shallow encoder, by controlling the norm of the weights or the code, but it becomes more subtle with a general pair of encoder and decoder.

We argue that this problem is a drawback of the reconstruction loss: when training an auto-encoder, we expect the function $\text{Dec} \circ \text{Enc}$ to be the identity function on the data manifold, but we still expect this function to extract meaningful

information. This means the function cannot be the identity on the whole space Ω. Thus, the reconstruction constraint, which leans towards learning the identity function is in opposition to another constraining term, that impedes the ability of the model to learn the identity function. Therefore, a careful balance of the terms is required, to prevent one or the other to dominate.
On the other hand, prediction does not suffer from this problem, the identity function is not a solution to prediction.

In our case, the prediction problem under the linearity constraint is formulated using a single loss function:

$$\mathcal{L}_\theta^{\text{pred}}(x, y) = ||\text{Dec}_\theta(2\text{Enc}_\theta(x^2) - \text{Enc}_\theta(x^1)) - y||_2^2 \tag{3.3}$$

where x are the past two frames and y is the future frame to predict. We see that in this case, the collapse of the system is not possible anymore.

3.1.2 Separation of *what* and *where*

Inspired by the philosophy revived by [Hinton et al., 2011], we introduce an *equivariant* representation of the frames. In that work, the concept of "capsule" units is introduced. An equivariant representation is learned by the capsules when the true latent states are provided to the network as implicit targets. Our work allows us to move to a more unsupervised setting in which the true latent states are not only unknown, but represent completely arbitrary qualities. This was made possible with two assumptions: (1) that temporally adjacent samples also correspond to neighbors in the latent space, (2) predictions of future samples can be formulated as *linear* operations in the latent space. In theory, the representation

learned by our method is very similar to the representation learned by the "capsules"; this representation has a locally stable *"what"* component and a locally linear, or equivariant *"where"* component. Theoretical properties of linearizing features were studied in [Cohen and Welling, 2014]. In other words, the linearized representation has the additional constraint that one part is constant during the video. This additional structure allows us to interpret the constant part of the code as the "what" (the global state of the world in the video), and the linear part as the "where" (the temporally dependent elements, such as the camera position). This property is achieved through a "phase-pooling" operator described in section 3.2.1. Another approach to generalize capsules will be explained in chapter 5.

3.2 Learning Linearized Representations

Our goal is to obtain a representation of each input sequence that varies linearly in time by transforming each frame *individually*. Furthermore, we assume that this transformation can be learned by a deep, feed forward encoder network Enc_θ. We assume that each video sequence in the dataset is parameterized by a temporal index t, such that a video can be written $x = \{..., x^{t-1}, x^t, x^{t+1}, ...\}$. In practice, however, we only consider triplets of frames, so we will always denote (x^1, x^2) the two input frames and $y = x^3$ the target frame, which corresponds to splitting each video samples into overlapping sequences of three frames. Correspondingly, we denote the z^1, z^2 and z^3 the codes for frames x^1, x^2 and y, such that $z^t = \text{Enc}_\theta(x^t)$, thus our goal is to train Enc_θ to produce a sequence z with minimal local curvature. As discussed in the previous section, we minimize the prediction error in the input space. The predicted frame is generated in two steps:

1. linearly extrapolation in code space to obtain a predicted code

$$\tilde{z^3} = \mathbf{a}[z^2 \;\; z^1]^\mathsf{T}; \tag{3.4}$$

2. decoding through Dec_θ, which generates the predicted frame

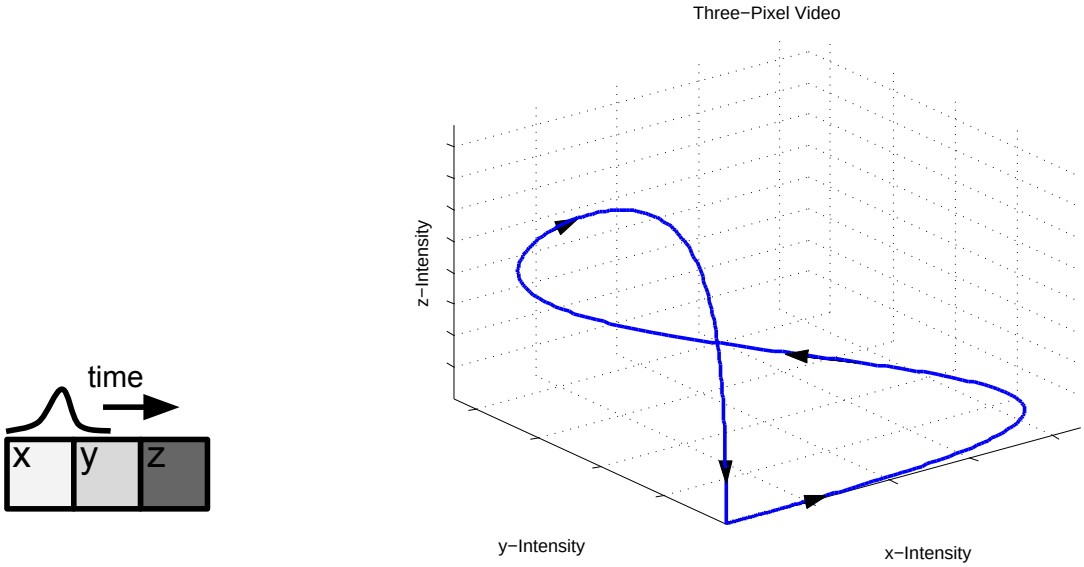
$$\tilde{y} = \mathrm{Dec}_\theta(\tilde{z^3}). \tag{3.5}$$

In practice, a constant speed linear extrapolation of z^1 and z^2 corresponds to $\mathbf{a} = [2, -1]$. The L^2 prediction error is minimized by jointly training the encoder and decoder networks, which assume unimodal target distribution. The case of multimodal targets is addressed in section 3.2.2.

Minimizing prediction error alone will not necessarily lead to low curvature trajectories in Z since the decoder is unconstrained; it may learn a many-to-one mapping which maps different codes to the same output image without forcing the codes to be equal. To prevent this, we add an explicit curvature penalty to the loss, corresponding to the cosine distance between $(z^2 - z^1)$ and $(z^3 - z^2)$. The complete loss to minimize is:

$$\mathcal{L}^{\text{linAE}} = \frac{1}{2}\left\|\mathrm{Dec}_\theta\left(\mathbf{a}\left[z^2 \;\; z^1\right]^\mathsf{T}\right) - y\right\|_2^2 - \lambda\frac{(z^2 - z^1)^\mathsf{T}(z^3 - z^2)}{\|z^2 - z^1\| \; \|z^3 - z^2\|} \tag{3.6}$$

where λ is a hyperparameter.



(a) (b)

Figure 3.1: A video generated by translating a Gaussian intensity bump (a) over a three pixel array $(\mathtt{x}, \mathtt{y}, \mathtt{z})$, and the corresponding manifold parametrized by time in three dimensional space (b).

3.2.1 Phase Pooling

Thus far we have assumed a generic architecture for Enc_θ and Dec_θ. We now consider custom architectures and operators that are particularly suitable for the task of linearization. To motivate the definition of these operators, consider a video generated by translating a Gaussian "intensity bump" over a three pixel region at constant speed. The video corresponds to a one dimensional manifold in three dimensional space, i.e. a curve parameterized by time (see Figure 3.1). Next, assume that some convolutional feature detector fires only when centered on the bump. Applying the *max*-pooling operator to the activations of the detector in this three-pixel region signifies the presence of the feature somewhere in this region (i.e. the "what"). Applying the *argmax* operator over the region returns the position (i.e. the "where") with respect to some local coordinate frame defined over the

pooling region. This position variable varies linearly as the bump translates, and thus parameterizes the curve in Figure 3.1b. These two channels, namely the *what* and the *where*, can also be regarded as generalized magnitude m and phase $\mathbf{p}$, corresponding to a factorized representation: the magnitude represents the active set of parameters, while the phase represents the set of local coordinates in this active set. We refer to the operator that outputs both the max and $argmax$ channels as the "phase-pooling" operator.

In this example, spatial pooling was used to linearize the translation of a fixed feature. More generally, the phase-pooling operator can locally linearize arbitrary transformations if pooling is performed not only spatially, but also across features in some topology.

In order to be able to back-propagate through $\mathbf{p}$, we define a soft version of the max and $argmax$ operators within each pool group. For simplicity, assume that the encoder has a fully convolutional architecture which outputs a set of feature maps, possibly of a different resolution than the input. Although we can define an arbitrary topology in feature space, for now assume that we have the familiar three-dimensional spatial feature map representation where each activation is a function $z(\mathtt{f}, \mathtt{x}, \mathtt{y})$, where $\mathtt{x}$ and $\mathtt{y}$ correspond to the spatial location, and $\mathtt{f}$ is the feature map index. Assuming that the feature activations are positive, we define our soft *"max-pooling"* operator for the k^{th} neighborhood N_k as:

$$m_k = \sum_{N_k} z(\mathtt{f}, \mathtt{x}, \mathtt{y}) \frac{e^{\beta z(\mathtt{f}, \mathtt{x}, \mathtt{y})}}{\sum_{N_k} e^{\beta z(\mathtt{f}', \mathtt{x}', \mathtt{y}')}} \approx \max_{N_k} z(\mathtt{f}, \mathtt{x}, \mathtt{y}), \tag{3.7}$$

where $\beta \geq 0$. Note that the fraction in the sum is a softmax operation (parametrized by β), which is positive and sums to one in each pooling region. The larger the

β, the closer it is to a unimodal distribution and therefore the better m_k approximates the *max* operation. On the other hand, if $\beta = 0$, Equation (3.7) reduces to *average-pooling*. Finally, note that m_k is simply the expected value of z (restricted to N_k) under the softmax distribution.

Assuming that the activation pattern within each neighborhood is approximately unimodal, we can define a soft versions of the *argmax* operator. The vector $\mathbf{p}_k$ approximates the local coordinates in the feature topology at which the max activation value occurred. Assuming that pooling is done volumetrically, that is, spatially *and* across features, $\mathbf{p}_k$ will have three components. In general, the number of components in $\mathbf{p}_k$ is equal to the dimension of the topology of our feature space induced by the pooling neighborhood. The dimensionality of $\mathbf{p}_k$ can also be interpreted as the *maximal intrinsic dimension of the data*. If we define a local standard coordinate system in each pooling volume to be bounded between -1 and +1, the soft "*argmax-pooling*" operator is defined by the vector-valued sum:

$$\mathbf{p}_k = \sum_{N_k} \begin{bmatrix} \mathtt{f} \\ \mathtt{x} \\ \mathtt{y} \end{bmatrix} \frac{e^{\beta z(\mathtt{f},\mathtt{x},\mathtt{y})}}{\sum_{N_k} e^{\beta z(\mathtt{f}',\mathtt{x}',\mathtt{y}')}} \approx \arg\max_{N_k} z(\mathtt{f},\mathtt{x},\mathtt{y}), \tag{3.8}$$

where the indices $\mathtt{f}, \mathtt{x}, \mathtt{y}$ take values from -1 to 1 in equal increments over the pooling region. Again, we observe that $\mathbf{p}_k$ is simply the expected value of $\begin{bmatrix} \mathtt{f} & \mathtt{x} & \mathtt{y} \end{bmatrix}^\mathsf{T}$ under the softmax distribution.

The phase-pooling operator acts on the output of the encoder, therefore it can simply be considered as the last encoding step. Correspondingly we define an *unpooling* operation as the first step of the decoder, which produces reconstructed activation maps by placing the magnitudes m at appropriate locations given by the phases $\mathbf{p}$, in a smooth manner (in our experiments, through linear interpolation).

Because the phase-pooling operator produces both magnitude and phase signals for each of the two input frames, it remains to define the predicted magnitude and phase of the third frame. In general, this linear extrapolation operator can be learned, however defining it manually allows us to place implicit priors on the magnitude and phase channels. We chose to keep the magnitude constant and linearly extrapolate the phase, so that the predicted magnitude and phase are defined as follows:

$$m^3 = \frac{m^2+m^1}{2} \tag{3.9}$$

$$\mathbf{p}^3 = 2\mathbf{p}^2 - \mathbf{p}^1 \tag{3.10}$$

Predicting the magnitude as the mean of the past imposes an implicit stability prior on m, i.e. the temporal sequence corresponding to the m channel should be stable between adjacent frames. The linear extrapolation of the phase variable imposes an implicit linear prior on $\mathbf{p}$. Thus such an architecture produces a factorized representation composed of a locally stable m and locally linearly varying $\mathbf{p}$. When phase-pooling is used curvature regularization is only applied to the $\mathbf{p}$ variables. The full prediction architecture is shown in Figure 3.2.

3.2.2 Addressing Uncertainty

Natural video can be inherently unpredictable; objects enter and leave the field of view, and out of plane rotations can also introduce previously invisible content. In this case, one acceptable value for the prediction could be the most likely outcome. However, if multiple outcomes are present in the training set then minimizing the L^2 distance to these multiple outcomes induces the network to predict

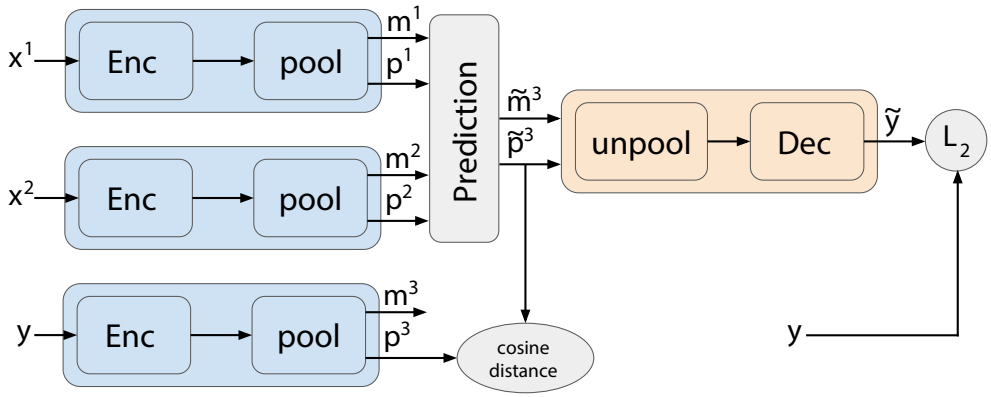


Figure 3.2: The basic linear prediction architecture with shared weight encoders.

the average outcome. In practice, this phenomena results in blurry predictions and may lead the encoder to learn a less discriminative representation of the input. To address this inherent unpredictability we introduce latent variables δ to the prediction architecture that are not deterministic functions of the input. These variables can be inferred *using the target* y in order to minimize the prediction L^2 error. The interpretation of these variables is that they explain all aspects of the predictions that are not captured by the encoder. For example, δ can be used to switch between multiple, equally likely predictions. Another interpretation of the latent variables is that although the distribution of the target given the input is multimodal, the distribution of the target given the input *and* the latent variable is unimodal, which allows us to use a MSE loss. In other words, $p_{\text{data}}(Y|X = x)$ is multimodal but we assume that $p_{\text{data}}(Y|X = x, Z = z)$ is a normal distribution. It is important to control the capacity of δ to prevent it from explaining the entire prediction on its own. Therefore δ is restricted to act only as a correction term in the code space output by the encoder. To further restrict the capacity of δ we enforce that $dim(\delta) \ll dim(z)$. More specifically, the δ-corrected code is defined

as:

$$\tilde{z}^3_\delta = z^2 + (W\delta) \odot \mathbf{a} \begin{bmatrix} z^2 & z^1 \end{bmatrix}^\mathsf{T} \tag{3.11}$$

Where W is a trainable matrix of size $dim(\delta) \times dim(z)$, and $\odot$ denotes the component-wise product. The entries of W are part of the trainable parameters θ. We define the loss given delta by

$$\mathcal{L}^{\text{linAE}}(\delta) = \|\text{Dec}_\theta(\tilde{z}^3_\delta) - y\|_2^2 - \lambda \frac{(z^2 - z^1)^\mathsf{T}(z^3 - z^2)}{\|z^2 - z^1\|\|z^3 - z^2\|}. \tag{3.12}$$

The final loss we wish to minimize is therefore

$$\mathcal{L}^{\text{linAE}\Delta} = \min_\delta \mathcal{L}^{\text{linAE}}(\delta). \tag{3.13}$$

During training, δ is inferred (using gradient descent) for each training sample by minimizing $\mathcal{L}^{\text{linAE}\Delta}(\delta)$ in Equation (3.13). The corresponding adjusted $\tilde{z}^3_\delta$ is then used during the update of θ via back-propagation. At test time δ can be sampled, assuming its distribution on the training set has been previously estimated (which may be a difficult problem in itself). Algorithm 3.1 details how the above loss is minimized using stochastic gradient descent.

When phase pooling is used we allow δ to only affect the phase variables in Equation (3.10), this further encourages the magnitude to be stable and places all the uncertainty in the phase.

3.3 Experiments

The following experiments evaluate the proposed feature learning architecture and loss. In the first set of experiments we train a shallow architecture on natural

Algorithm 3.1 Minibatch stochastic gradient descent training for video prediction with uncertainty. The number of δ-gradient descent steps k is treated as a hyper-parameter.

for number of training epochs **do**
  Sample a mini-batch of temporal triplets $\{x^1, x^2, y\}$
  Set $\delta_0 = 0$
  Forward propagate x^1, x^2 through Enc_θ and obtain the codes z^1, z^2
  Compute $\tilde{z}^3_{\delta_0}$ and the prediction $\tilde{y}_0 = \text{Dec}_\theta(\tilde{z}^3_{\delta_0})$
  for i =1 to k **do**
    Compute the L^2 prediction error
    Back propagate the error through Dec_θ to compute the gradient $\frac{\partial \mathcal{L}^{\text{linAE}(\delta_{\text{i}-1})}}{\partial \delta_{i-1}}$
    Update $\delta_i = \delta_{i-1} - \alpha \frac{\partial \mathcal{L}^{\text{linAE}(\delta_{\text{i}-1})}}{\partial \delta_{i-1}}$
    Compute $\tilde{z}^3_{\delta_i} = z^2 + (W\delta_i) \odot \mathbf{a}\,[z^2 \quad z^1]^\mathsf{T}$
    Compute $\tilde{y}_i = \text{Dec}_\theta(\tilde{z}^3_{\delta_i})$
  end for
  Back propagate the final prediction error to compute $\frac{\partial \mathcal{L}^{\text{linAE}(\delta_{\text{k}})}}{\partial \theta}$
  Update $\theta = \theta - \lambda \frac{\partial \mathcal{L}^{\text{linAE}(\delta_{\text{k}})}}{\partial \theta}$
end for

data and visualize the learned features to gain a basic intuition. In the second set of experiments we train a deep architecture on simulated movies generated from the NORB dataset. By generating frames from interpolated and extrapolated points in code space we show that a linearized representation of the input is learned. Finally, we explore the role of uncertainty by training on only partially predictable sequences, we show that our latent variable formulation can account for this uncertainty enabling the encoder to learn a linearized representation even in this setting.

3.3.1 Shallow Architecture Trained on Natural Data

To gain an intuition for the features learned by a phase-pooling architecture let us consider an encoder architecture comprised of the following stages: convo-

lutional filter bank, rectifying point-wise nonlinearity, and phase-pooling. The decoder architecture is comprised of an un-pooling stage followed by a convolutional filter bank. This architecture was trained on simulated 32×32 movie frames taken from YouTube videos [Goroshin et al., 2015a]. Each frame triplet is generated by transforming still frames with a sequence of three rigid transformations (translation, scale, rotation). More specifically let A be a random rigid transformation parameterized by τ, and let x denote a still image reshaped into a column vector, the generated triplet of frames is given by $\{x^1 = A_{\tau=\frac{1}{3}}x, x^2 = A_{\tau=\frac{2}{3}}x, y = A_{\tau=1}x\}$. Two variants of this architecture were trained, described as Shallow Architecture 1 and 2 in Table 3.1. In Shallow Architecture 1, phase pooling is performed spatially in non-overlapping groups of 4×4 and across features in a one-dimensional topology consisting of non-overlapping groups of four. Each of the 16 pool-groups produce a code consisting of a scalar m and a three-component $\mathbf{p} = [p_{\mathtt{f}}\ p_{\mathtt{x}}\ p_{\mathtt{y}}]^{\mathsf{T}}$ (corresponding to two spatial and one feature dimensions); thus the encoder architecture produces a code of size $16 \times 4 \times 8 \times 8$ for each frame. The corresponding filters whose activations were pooled together are laid out horizontally in groups of four in Figure 3.3a. Note that each group learns to exhibit a strong ordering corresponding to the linearized variable p_f. Because global rigid transformations can be locally well approximated by translations, the features learn to parameterize local translations. In effect the network learns to linearize the input by tracking common features in the video sequence. Unlike the spatial phase variables, p_f can linearize sub-pixel translations. Next, Shallow Architecture 2 described in Table 3.1 was trained on natural movie patches with the natural motion present in the real videos. The architecture differs in only in that pooling across features is done with overlap (groups of 4, stride of 2). The resulting decoder filters are displayed

Shallow Architecture 1	
Encoder	$\underset{32\times32}{3} \xrightarrow[9]{\text{convp}} \xrightarrow{\text{ReLU}} \underset{32\times32}{64} \xrightarrow[4/4]{\text{PhasePool}} \underset{8\times8}{16}$
Predictor	Magnitude m: average; Phase $\mathbf{p}$: linear extrapolation
Decoder	$\underset{8\times8}{16} \xrightarrow{\text{UnPool}} \underset{32\times32}{64} \xrightarrow[9]{\text{convp}} \underset{32\times32}{3}$

Shallow Architecture 2	
Encoder	$\underset{32\times32}{3} \xrightarrow[9]{\text{convp}} \xrightarrow{\text{ReLU}} \underset{32\times32}{64} \xrightarrow[4/2]{\text{PhasePool}} \underset{16\times16}{32}$
Predictor	Magnitude m: average; Phase $\mathbf{p}$: linear extrapolation
Decoder	$\underset{16\times16}{32} \xrightarrow{\text{UnPool}} \underset{32\times32}{64} \xrightarrow[9]{\text{convp}} \underset{32\times32}{3}$

Deep Architecture 1	
Encoder	$\underset{32\times32}{1} \xrightarrow[9]{\text{conv}} \xrightarrow{\text{ReLU}} \underset{24\times24}{16} \xrightarrow[9]{\text{conv}} \xrightarrow{\text{ReLU}} \underset{16\times16}{32} \xrightarrow{\text{FC}} \xrightarrow{\text{ReLU}} 4096$
Predictor	None
Decoder	$2*4096 \xrightarrow{\text{concat}} 8192 \xrightarrow{\text{FC}} \xrightarrow{\text{ReLU}} 8192 \xrightarrow{\text{reshape}} \underset{16\times16}{32} \xrightarrow[8]{\text{pad}} \xrightarrow[9]{\text{conv}} \xrightarrow{\text{ReLU}} \underset{24\times24}{16} \xrightarrow[8]{\text{pad}} \xrightarrow[9]{\text{conv}} \underset{32\times32}{1}$

Deep Architecture 2	
Encoder	$\underset{32\times32}{1} \xrightarrow[9]{\text{conv}} \xrightarrow{\text{ReLU}} \underset{24\times24}{16} \xrightarrow[9]{\text{conv}} \xrightarrow{\text{ReLU}} \underset{16\times16}{32} \xrightarrow{\text{FC}} \xrightarrow{\text{ReLU}} 4096$
Predictor	Linear extrapolation
Decoder	$4096 \xrightarrow{\text{FC}} \xrightarrow{\text{ReLU}} 8192 \xrightarrow{\text{reshape}} \underset{16\times16}{32} \xrightarrow[8]{\text{pad}} \xrightarrow[9]{\text{conv}} \xrightarrow{\text{ReLU}} \underset{24\times24}{16} \xrightarrow[8]{\text{pad}} \xrightarrow[9]{\text{conv}} \underset{32\times32}{1}$

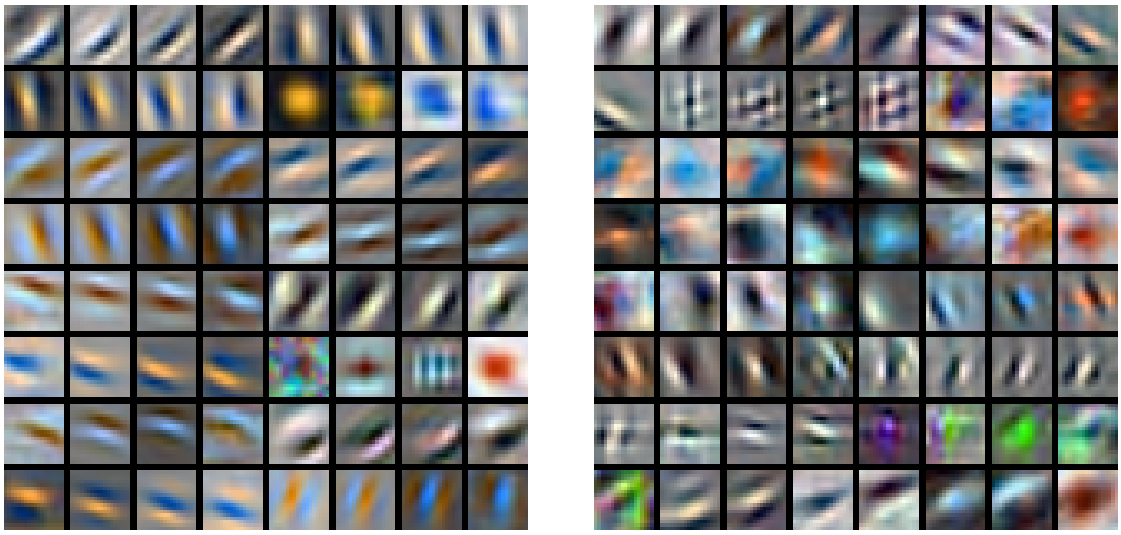
Deep Architecture 3	
Encoder	$\underset{32\times32}{1} \xrightarrow[9]{\text{conv}} \xrightarrow{\text{ReLU}} \underset{24\times24}{16} \xrightarrow[9]{\text{conv}} \xrightarrow{\text{ReLU}} \underset{16\times16}{32} \xrightarrow{\text{FC}} \xrightarrow{\text{ReLU}} 4096 \xrightarrow{\text{reshape}} \underset{8\times8}{64} \xrightarrow[8/8]{\text{PhasePool}} \underset{1\times1}{64}$
Predictor	Magnitude m: average; Phase $\mathbf{p}$: linear extrapolation
Decoder	$\underset{1\times1}{64} \xrightarrow{\text{UnPool}} \underset{8\times8}{64} \xrightarrow{\text{reshape}} 4096 \xrightarrow{\text{FC}} \xrightarrow{\text{ReLU}} 8192 \xrightarrow{\text{reshape}} \underset{16\times16}{32} \xrightarrow[8]{\text{pad}} \xrightarrow[9]{\text{conv}} \xrightarrow{\text{ReLU}} \underset{24\times24}{16} \xrightarrow[8]{\text{pad}} \xrightarrow[9]{\text{conv}} \underset{32\times32}{1}$

Table 3.1: Summary of architectures used for experiments.

in Figure 3.3b. Note that pooling with overlap introduces smoother transitions between the pool groups. Although some groups still capture translations, more complex transformations are learned from natural movies.

3.3.2 Deep Architecture trained on NORB

In the next set of experiments we trained deep feature hierarchies that have the capacity to linearize a richer class of transformations. To evaluate the properties of the learned features in a controlled setting, the networks were trained on



(a) Shallow Architecture 1: non overlapping groups of 4 trained on synthetic videos

(b) Shallow Architecture 2: groups of 4 overlapping by 2 trained on natural videos

Figure 3.3: Decoder filters learned by shallow phase-pooling architectures.

simulated videos generated using the NORB dataset rescaled to 32×32 to reduce training time. The simulated videos are generated by tracing constant speed trajectories with random starting points in the two-dimensional latent space of pitch and azimuth rotations. In other words, the models are trained on triplets of frames ordered by their rotation angles. As before, the models are trained to predict the third frame given two input frames. Recall that prediction is a proxy for learning linearized feature representations. One way to evaluate the linearization properties of the learned features is to linearly interpolate (or extrapolate) new codes and visualize the corresponding images via forward propagation through the decoder. This simultaneously tests the encoder's capability to linearize the input and the decoder's (generative) capability to synthesize images from the linearized codes. In order to perform these tests we must have an *explicit* code representation, which is not always available. For instance, consider a simple scheme in which a generic

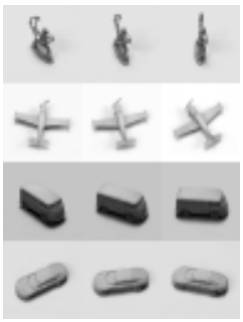


Figure 3.4: Input samples used for testing.

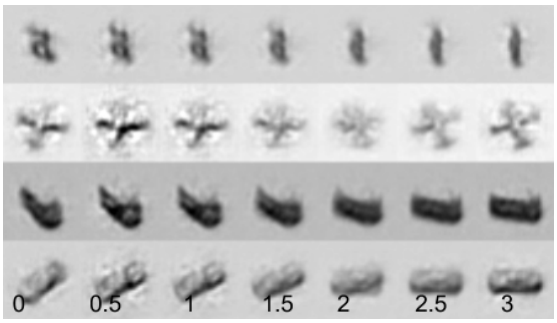


Figure 3.5: Linear interpolation in code space using the baseline Siamese-encoder network (Deep Architecture 1).

deep network is trained to predict the third frame from the concatenated input of two previous frames. Such a network does not even provide an explicit feature representation for evaluation. A simple baseline architecture that affords this type of evaluation is a Siamese encoder followed by a decoder, this exactly corresponds to our proposed architecture with the linear prediction layer removed. Such an architecture is equivalent to learning the weights of the linear prediction layer of the model shown in Figure 3.2. In the following experiment we evaluate the effects of: (1) fixing v.s. learning the linear prediction operator, (2) including the phase pooling operation, and (3) including explicit curvature regularization (second term in Equation (3.6)).

Let us first consider Deep Architecture 1 summarized in Table 3.1. In this architecture a Siamese encoder produces a code of size 4096 for each frame. The codes corresponding to the two frames are concatenated together and propagated to the decoder. In this architecture the first linear layer of the decoder can be interpreted as a learned linear prediction layer. Figure 3.4 shows three frames from the test set corresponding to temporal indices 1, 2 and 3 respectively. Figure 3.5 shows the generated frames corresponding to interpolated codes at temporal

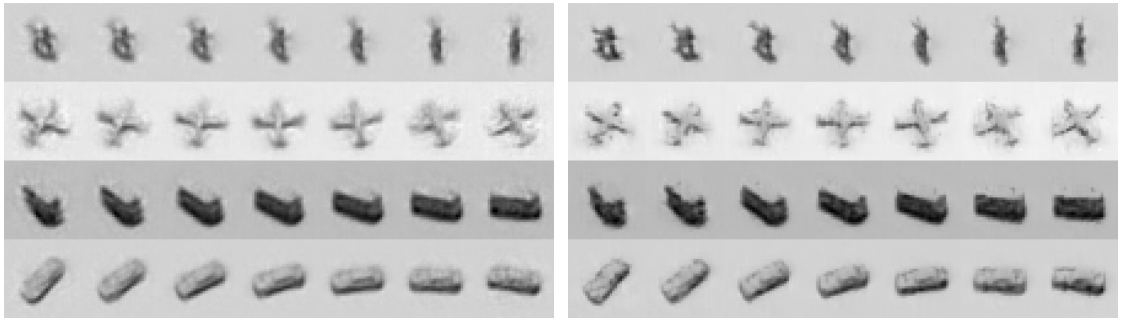
indices: $\{0, \frac{1}{2}, 1, \frac{3}{2}, 2, \frac{5}{2}, 3\}$. The images were generated by propagating the corresponding codes through the decoder. Codes corresponding to non-integer temporal indices were obtained by linearly interpolating in code space.

Deep Architecture 2 differs from Deep Architecture 1 in that it generates the predicted code via a fixed linear extrapolation in code space. The extrapolated code is then fed to the decoder that generates the predicted image. Note that the fully connected stage of the decoder has half as many free parameters compared to the previous architecture. This architecture is further restricted by propagating only the predicted code to the decoder. For instance, unlike in Deep Architecture 1, the decoder cannot copy any of the input frames to the output. The generated images corresponding to this architecture are shown in Figure 3.6a. These images more closely resemble images from the dataset. Furthermore, Deep Architecture 2 achieves a lower L^2 prediction error than Deep Architecture 1.

Finally, Deep Architecture 3 uses phase-pooling in the encoder, and "un-pooling" in the decoder. This architecture makes use of phase-pooling in a two-dimensional feature space arranged on an 8×8 grid. The pooling is done in a single group over all the fully-connected features producing a feature vector of dimension 192 (64×3) compared to 4096 in previous architectures. Nevertheless this architecture achieves the best overall L^2 prediction error and generates the most visually realistic images (Figure 3.6b).

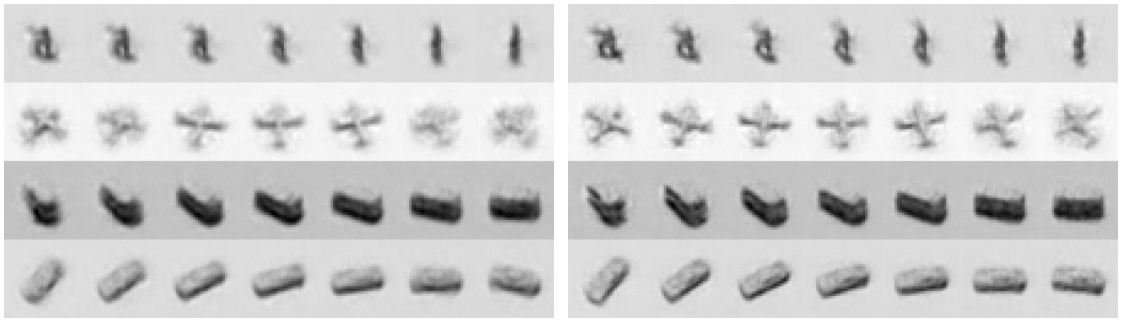
3.3.3 Uncertainty

In this subsection, we compare the representation learned by minimizing the loss in Equation (3.6), i.e. without latent variables, to Equation (3.13), with latent variables. Uncertainty is simulated by generating triplet sequences where the third



(a) Deep Architecture 2 (b) Deep Architecture 3

Figure 3.6: Linear interpolation in code space learned by our models: Comparison between Deep Architecture 2 (no phase-pooling, no curvature regularization) and Deep Architecture 3 (with phase pooling and curvature regularization).



(a) Minimizing Equation (3.6): no latent variables (b) Minimizing Equation (3.13): using latent variables

Figure 3.7: Interpolation results on data with uncertainty, with and without latent variables. Both figures are using Deep Architecture 3.

frame is skipped randomly with equal probability, determined by Bernoulli variable s. For example, the sequences corresponding to models with rotation angles $0°, 20°, 40°$ and $0°, 20°, 60°$ are equally likely. Using the loss without latent variables (Equation (3.6)) with Deep Architecture 3 results in the images displayed in Figure 3.7a. The interpolations are blurred due to the averaging effect discussed in Subsection 3.2.2. On the other hand, Figure 3.7 shows the results when minimizing Equation (3.13). It partially recovers the sharpness of Figure 3.6b. For this experiment, we used a three-dimensional, real-valued δ. Moreover training a linear predictor to infer binary variable s from δ (after training) results in a 94% test set accuracy. This suggests that δ does indeed capture the uncertainty in the data.

3.4 Conclusion

The prediction objective works around several problems of auto-encoders. We presented a linearizing regularizer that is able to identify the time component of videos and disentangle it from the other factors. We introduced a phase pooling and unpooling that is able to further separate the "what" and the "where" through a specific architecture. Finally, the latent variables δ are able to capture some of the unpredictability of the data. However, the predictions obtained with this architecture are still blurry and do not scale well to larger datasets. One possible explanation is the failure of δ is correctly capture the uncertainty without explaining the entire prediction. The next chapter brings new solutions to cope with the problem of uncertainty in the context of video prediction.

Chapter 4

Deep multi-scale video prediction beyond mean square error

4.1 Introduction

This chapter focuses on alternatives to the L^2 distance as a training criterion for prediction. Using latent variables in order to explain uncertainty away is a general technique that has the theoretical ability to address the problem of multimodal target distributions. However, there exist different techniques in order to train a model to use latent variables, and gradient descent, as presented in Chapter 3, is not without problems. Inferring the variables requires k forward and backward passes in the model, which makes the training $k + 1$ times slower. Moreover, gradient descent on the latent variables δ will infer the variables that minimize the loss, regardless of whether or not the information contained in δ is already contained in the input or not. Therefore, it is important to carefully restrict the capacity of δ, which may be difficult in general, non-synthetic setups.

In this chapter, we will see how Generative Adversarial Networks (GANs) [Goodfellow et al., 2014] can handle the same problem in a different way. This chapter also addresses the problem of lack of sharpness in the predictions, by assessing different loss functions, including generative adversarial training, and a new auxiliary loss based on the image gradients, designed to preserve the sharpness of the frames. Combining these two losses produces the most visually satisfying results.

This chapter is organized as follows: section 4.2 describes the different model architectures: simple, multi-scale, adversarial, and presents the Gradient Difference Loss (GDL) function. The experimental section 4.3 compares the proposed architectures and losses on video sequences from the Sports1m dataset [Karpathy et al., 2014], UCF101 [Soomro et al., 2012] and NYUDepth [Nathan Silberman and Fergus, 2012]. We evaluate the quality of image generation by computing similarity and sharpness measures, and showing instances of future frame prediction. Finally, section 4.4 discusses the remaining issues and possible applications of video prediction.

4.2 Models

Let $y = \{y^1, ..., y^n\}$ be a sequence of frames to predict from input frames $x = \{x^1, ..., x^m\}$ in a video sequence. Our approach is based on a convolutional network [LeCun et al., 1998], alternating convolutions and Rectified Linear Units (ReLU) [Nair and Hinton, 2010].

Such a network G, displayed in Figure 4.1, can be trained to predict one or several concatenated frames y from the concatenated frames x by minimizing a

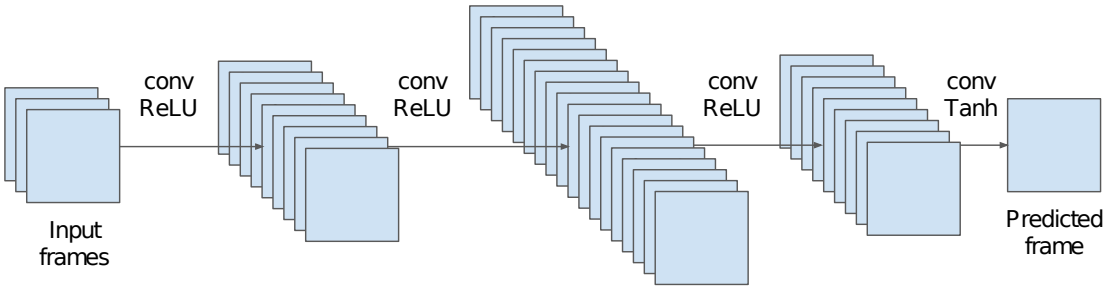


Figure 4.1: A basic convolutional network for next frame prediction. Note that this is just an illustration of the structure and the actual number of layers and feature maps differs from experiment to experiment.

distance, for instance L^p with $p = 1$ or $p = 2$, between the predicted frame and the true frame:

$$\mathcal{L}^{\mathrm{p}}(x, y) = L^p(G(x), y) = \|G(x) - y\|_p^p. \tag{4.1}$$

However, such a network has at least two major flaws:

Aperture problem: convolutions only account for short-range dependencies, limited by the size of their kernels. Using pooling would only be part of the solution since the output has to be of the same resolution as the input. There exist a number of ways to avoid the loss of resolution brought about by pooling or subsampling while preserving long-range dependencies. The simplest and most obvious one is to have no pooling or subsampling but many convolution layers [Jain et al., 2007]. Another method is to use connections that "skip" the pooling/unpooling pairs, to preserve the high-frequency information [Long et al., 2015, Dosovitskiy et al., 2014, Ronneberger et al., 2015]. Finally, we can combine multiple scales linearly as in the reconstruction process of a Laplacian pyramid [Denton et al., 2015]. This is the approach we use here.

Blurry predictions: using an L^2 loss, and to a lesser extent L^1, corresponds to parametrizing unimodal distributions as the output of the network (see Chapter 2). It necessarily produces blurry predictions with natural videos, increasingly worse when predicting further in the future. If the probability distribution for an output pixel has two equally likely modes v_1 and v_2, the value $v_{avg} = (v_1 + v_2)/2$ minimizes the L^2 loss over the data, even if v_{avg} has very low probability. In the case of an L^1 norm, this effect diminishes, but does not disappear, as the output value would be the median of the set of equally likely values.

4.2.1 Multi-scale network

We tackle the aperture problem by making the model multi-scale. A multi-scale version of the model is composed of N_{scales} models, each of them with a structure close to the model depicted in Figure 4.1. Let $s_1, \ldots, s_{N_{\text{scales}}}$ be the sizes of the inputs of our networks. Typically, in our experiments, we set $s_1 = 4 \times 4$, $s_2 = 8 \times 8$, $s_3 = 16 \times 16$, $s_4 = 32 \times 32$ and, when applicable, $s_5 = 64 \times 64$. Let u_k be the upscaling operator toward size s_k. Let x_k^i, y_k^i denote the downscaled versions of x^i and y^i of size s_k. We define the model G_k that takes x_k and $\hat{y}_{k-1}$ as inputs and learns to predict y_k. This model is illustrated in Figure 4.2, and has a special structure: the input $\hat{y}_{k-1}$ is first upsampled (using a fixed bilinear upsampling) and then concatenated to x_k. Additionally, the output of the convolutional network is not trained to be y_k, but instead $y_k - u_k(y_{k-1})$. This "residual" form to the network is inspired from Laplacian pyramids, and makes the training procedure easier.

Consequently, the network makes a series of predictions, starting from the lowest resolution, and uses the prediction of size s_k as a starting point to make

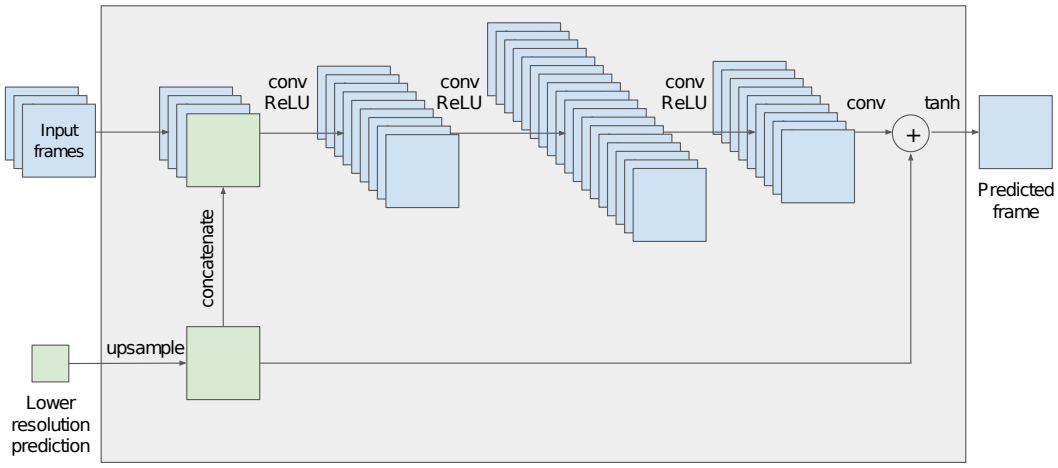


Figure 4.2: A basic block G_k of the multi-scale video prediction structure. Note that this is just an illustration of the structure and the actual number of layers and feature maps differs from experiment to experiment.

the prediction of size s_{k+1}. At the lowest scale s_1, the network takes only x_1 as an input, since there is no lower scale. This architecture is illustrated in Figure 4.3, and the specific details are given in Section 4.3. We denote θ_G the set of trainable parameters of the generator G.

Despite the multi-scale architecture, the problem of blurry predictions remains. We address this problem in the next two sections, through an adversarial strategy and the image Gradient Difference Loss (GDL).

4.2.2 Adversarial training

Generative Adversarial Networks were introduced by [Goodfellow et al., 2014], and are used to generate image patches from random noise using two networks trained simultaneously. In that work, the authors propose to use a discriminative network D to estimate the probability that a sample comes from the dataset instead of being produced by a generative model G. The two models are simultaneously trained so that G learns to generate frames that are hard to classify by D, while

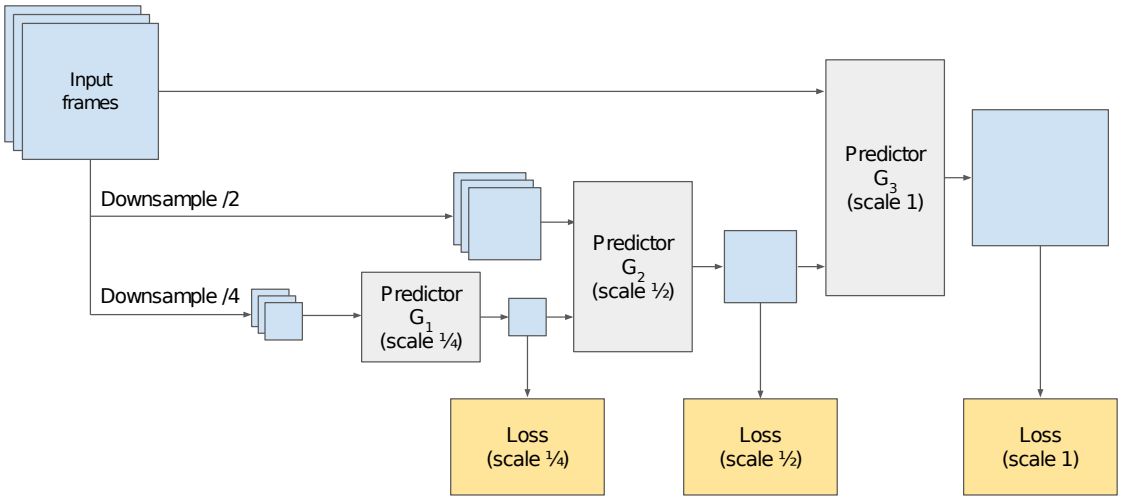


Figure 4.3: The full multi-scale video prediction generator G. In this illustration, we set $N_{\text{scales}} = 3$ for clarity, but in the experiments we typically have N_{scales} equal to 4 or 5.

D learns to discriminate the frames generated by G. Ideally, when G is trained, it should not be possible for D to perform better than chance. For more details about Generative Adversarial Networks, refer to section 2.1.3.2.

We adapted this approach for the purpose of frame prediction, which constitutes to our knowledge the first application of adversarial training to video prediction. The generative model G is typically the one described in the previous section. The discriminative model D takes a sequence of frames, and is trained to predict the probability that the last frames of the sequence are generated by G. Note that only the last frames are either real of generated by G, the rest of the sequence is always from the dataset. This allows the discriminative model to make use of temporal information, so that G learns to produce sequences that are temporally coherent with its input.

Conditional adversarial training can be seen as another method to train a predictive network in the presence of latent variables (as presented in Chapter 3),

if we consider the input noise ϵ as latent variables. Indeed, the theory of GANs [Goodfellow et al., 2014] states that after convergence, a generator $G(x, \epsilon)$ generates samples from $p(Y|X = x)$ when ϵ is sampled from its fixed distribution. In other words, while the inference method presented in Chapter 3 takes a pair (x, y) and finds the optimal z that explains y from x, conditional GANs take a pair (x, z), where $z = \epsilon$, and make the generator produce a realistic y. There is no explicit target for y anymore, but the discriminator network is able to produce gradients nonetheless. Moreover, the discriminator has the theoretical ability to make the distribution of $G(x, \epsilon)$ follow $p(Y|X = x)$ even when it is multimodal. In practice, this property is harder to obtain and the Mode Collapse problem can arise when the generator chooses one mode of $p(Y|X = x)$ and only generates samples in this mode. This issue is further discussed in section 4.4.

In this work, the discriminative model D is matching the multi-scale generator G. Every prediction $\hat{y}_k$ goes to a different discriminator D_k, so the whole discriminator D is composed of N_{scales} discriminators D_k, each of them produces a single scalar output. The training of the pair (G, D) consists of two alternated steps, described below. For the sake of clarity, we assume that we use pure stochastic steps (minibatches of size 1), but there is no difficulty to generalize the algorithm to minibatches of size M by summing the losses over the samples. The trainable parameters of D are denoted θ_D.

Training D: Let (x, y) be a sample from the dataset, and ϵ a noise vector, for instance $\epsilon \sim \mathcal{N}(0, 1)$. Note that x (respectively y) is a sequence of m (respectively n) frames. We train D to classify the input (x, y) into class 1 and the input $(x, G(x, \epsilon))$ into class 0. More precisely, for each scale k, we perform one SGD

(or Adam [Kingma and Ba, 2014]) iteration of D_k while keeping the weights of G fixed. It is trained with the target 1 for the datapoint (x_k, y_k), and the target 0 for $(x_k, G_k(x, \epsilon))$. Note that in this section in order to keep the notations simple, we denote $G_k(x, \epsilon)$ the output of the generator G_k (instead of $G_k(x_k, \hat{y}_{k-1}, \epsilon)$). The loss function we use to train D is

$$\mathcal{L}_D^{\text{adv}}(x, y, \epsilon) = \sum_{k=1}^{N_{\text{scales}}} L^{bce}(D_k(x_k, y_k), 1) + L^{bce}(D_k(x_k, G_k(x, \epsilon)), 0). \tag{4.2}$$

L^{bce} is the binary cross-entropy loss, defined as

$$L^{bce}(y, \hat{y}) = -\sum_i \hat{y}_i \log(y_i) + (1 - \hat{y}_i) \log(1 - y_i) \tag{4.3}$$

where y_i takes its values in $\{0, 1\}$ and $\hat{y}_i$ in $[0, 1]$.

Training G: Let (x, y) be a data sample and ϵ a noise vector (they can be the same that were used to train D). While keeping the weights of D fixed, we perform one SGD (or Adam) step on G to minimize the adversarial loss:

$$\mathcal{L}_G^{\text{adv}}(x, y, \epsilon) \quad = \quad \sum_{k=1}^{N_{\text{scales}}} L^{bce}(D_k(x_k, G_k(x, \epsilon)), 1). \tag{4.4}$$

Note that although only G_k appears in this equation, the gradients are actually back-propagated through all $G_{k'}$ for $k' \leq k$. Minimizing this loss means that the generative model G is making the discriminative model D as "confused" as possible, in the sense that D will not classify the prediction correctly. However, in practice, minimizing this loss alone can lead to instability, in particular early in the training when the generator produces bad predictions, the gradients to train D are

not informative and the whole system may never get close to a Nash equilibrium. To address this problem, in addition to the adversarial loss, in some experiments we use an auxiliary loss in pixel space, for instance $\mathcal{L}^{\text{p}}$. Although this loss will likely make the predictions blurrier, it helps stabilize the training (and can be tuned down during training). The generator G is therefore trained to minimize $\lambda_{adv}\mathcal{L}_G^{\text{adv}} + \lambda_p\mathcal{L}^{\text{p}}$. There is therefore a tradeoff to adjust, by the mean of the hyper-parameters λ_{adv} and λ_p, between sharp predictions due to the adversarial principle, and stability brought by the second term. It may help to use a schedule for λ_p, reducing it over time. This process is summarized in Algorithm 4.1, with minibatches of size M.

Algorithm 4.1 Training via a conditional generative adversarial network for next frame generation (with SGD and no scheduling).

Set the learning rates ρ_D and ρ_G, and relative weights λ_{adv}, λ_p.
while not converged **do**
  Get M data samples $(x, y) = (x^{(1)}, y^{(1)}), \ldots, (x^{(M)}, y^{(M)})$
  Update the discriminator D:
  Sample $\epsilon \sim \mathcal{N}(0, 1)$
  $\theta_D = \theta_D - \rho_D \sum_{i=1}^{M} \frac{\partial \mathcal{L}_D^{\text{adv}}(x^{(i)}, y^{(i)}, \epsilon)}{\partial \theta_D}$
  Update the generator G:
  $\theta_G = \theta_G - \rho_G \sum_{i=1}^{M} \left(\lambda_{adv} \frac{\partial \mathcal{L}_G^{\text{adv}}(x^{(i)}, y^{(i)}), \epsilon}{\partial \theta_G} + \lambda_p \frac{\partial \mathcal{L}^{\text{p}}\left(G(x^{(i)}, \epsilon), y^{(i)}\right)}{\partial \theta_G} \right)$
end while

4.2.3 Image Gradient Difference Loss (GDL)

Another strategy to sharpen the image prediction is to directly penalize the differences of image gradient predictions in the generative loss function. We define a new loss function, the Gradient Difference Loss (GDL), that can be combined with the $\mathcal{L}^{\text{p}}$ and/or adversarial loss function. The GDL function between the

ground truth image y, and the prediction $G(x, \epsilon) = \hat{y}$ is given by

$$\mathcal{L}^{\text{gdl}}(x, y) = L^{gdl}(\hat{y}, y) = \sum_{i,j} \left| |y_{i,j} - y_{i-1,j}| - |\hat{y}_{i,j} - \hat{y}_{i-1,j}| \right|^{\alpha} + \left| |y_{i,j-1} - y_{i,j}| - |\hat{y}_{i,j-1} - \hat{y}_{i,j}| \right|^{\alpha}, \quad (4.5)$$

where α is an integer greater or equal to 1, and $|.|$ denotes the absolute value function. To the best of our knowledge, the closest related work to this idea is the work of [Mahendran and Vedaldi, 2015], using a total variation regularization to generate images from learned features. Our GDL is fundamentally different: in [Mahendran and Vedaldi, 2015], the total variation takes only the reconstructed frame in input, whereas our loss penalizes gradient differences between the prediction and the true output. Second, we chose the simplest possible image gradient by considering the neighbor pixel intensities differences, rather than adopting a more sophisticated norm on a larger neighborhood, for the sake of keeping the training time low. This loss can be used as an auxiliary pixel loss as defined in section 4.2.2, instead of (or in addition to) $\mathcal{L}^{\text{p}}$.

4.2.4 Combining losses

In our experiments, we combine the losses previously defined with different weights. The final loss is:

$$\mathcal{L}(x, y) = \lambda_{adv}\mathcal{L}_G^{\text{adv}}(x, y) + \lambda_p\mathcal{L}^{\text{p}}(x, y) + \lambda_{gdl}\mathcal{L}^{\text{gdl}}(x, y). \quad (4.6)$$

The hyperparameters λ_{adv}, λ_p and λ_{gdl} can also follow a schedule, and tuning down λ_p and λ_{gdl} over time tends to produce sharper predictions, while preserving the

Generators	
G_1	$12 \xrightarrow[3]{\text{convp, ReLU}} 128 \xrightarrow[3]{\text{convp, ReLU}} 256 \xrightarrow[3]{\text{convp, ReLU}} 128 \xrightarrow[3]{\text{convp, Tanh}} 3$
G_2	$12 \xrightarrow[5]{\text{convp, ReLU}} 128 \xrightarrow[3]{\text{convp, ReLU}} 256 \xrightarrow[3]{\text{convp, ReLU}} 128 \xrightarrow[5]{\text{convp, Tanh}} 3$
G_3	$12 \xrightarrow[5]{\text{convp, ReLU}} 128 \xrightarrow[3]{\text{convp, ReLU}} 256 \xrightarrow[3]{\text{convp, ReLU}} 512 \xrightarrow[3]{\text{convp, ReLU}} 256 \xrightarrow[3]{\text{convp, ReLU}} 128 \xrightarrow[5]{\text{convp, Tanh}} 3$
G_4	$12 \xrightarrow[7]{\text{convp, ReLU}} 128 \xrightarrow[5]{\text{convp, ReLU}} 256 \xrightarrow[5]{\text{convp, ReLU}} 512 \xrightarrow[5]{\text{convp, ReLU}} 256 \xrightarrow[5]{\text{convp, ReLU}} 128 \xrightarrow[7]{\text{convp, Tanh}} 3$

Discriminators	
D_1	$15 \xrightarrow[3]{\text{conv, ReLU}} 64 \xrightarrow{\text{FC, ReLU}} 512 \xrightarrow{\text{FC, ReLU}} 256 \xrightarrow{\text{FC, } \sigma} 1$
D_2	$15 \xrightarrow[3]{\text{conv, ReLU}} 64 \xrightarrow[3]{\text{conv, ReLU}} 128 \xrightarrow[3]{\text{conv, ReLU}} 128 \xrightarrow{\text{FC, ReLU}} 1024 \xrightarrow{\text{FC, ReLU}} 512 \xrightarrow{\text{FC, } \sigma} 1$
D_3	$15 \xrightarrow[5]{\text{conv, ReLU}} 128 \xrightarrow[5]{\text{conv, ReLU}} 256 \xrightarrow[5]{\text{conv, ReLU}} 256 \xrightarrow{\text{FC, ReLU}} 1024 \xrightarrow{\text{FC, ReLU}} 512 \xrightarrow{\text{FC, } \sigma} 1$
D_4	$15 \xrightarrow[7]{\text{conv, ReLU}} 128 \xrightarrow[7]{\text{conv, ReLU}} 256 \xrightarrow[5]{\text{conv, ReLU}} 512 \xrightarrow[5]{\text{conv, MaxPool, ReLU}} 512 \xrightarrow{\text{FC, ReLU}} 1024 \xrightarrow{\text{FC, ReLU}} 512 \xrightarrow{\text{FC, } \sigma} 1$

Table 4.1: Frame prediction Model 1 architectures.

stability benefits in early training.

4.3 Experiments

We now provide a quantitative evaluation of the quality of our video predictions on the UCF101 [Soomro et al., 2012], Sports1m [Karpathy et al., 2014] and NYUDepth (version 2) [Nathan Silberman and Fergus, 2012] datasets. We train the models to predict a single frame, but we test the stability of the predictions by applying the model recursively to predict more than one frame.

This experimental section presents two different models, called Model 1 and Model 2. Model 1 was trained on 32×32 video patches from the Sports1m dataset, after making sure they show enough movement, quantified by the L^2 distance between successive frames. We test this model on Sports1m and UCF101 (after fine tuning it on 64×64 patches from UCF101). Model 2 is trained and tested on 64×64 patches from NYUDepth.

Generators	
G_1	$9 \xrightarrow[5]{\text{convp}} \xrightarrow[\text{BN}]{\text{ReLU}} 32 \xrightarrow[5]{\text{convp}} \xrightarrow[\text{BN}]{\text{ReLU}} 128 \xrightarrow[5]{\text{convp}} \xrightarrow[\text{BN}]{\text{ReLU}} 256 \xrightarrow[5]{\text{convp}} \xrightarrow[\text{BN}]{\text{ReLU}} 128 \xrightarrow[5]{\text{convp}} \xrightarrow[\text{BN}]{\text{ReLU}} 32 \xrightarrow[5]{\text{convp}} \xrightarrow{\text{Tanh}} 3$
G_2	$9 \xrightarrow[5]{\text{convp}} \xrightarrow[\text{BN}]{\text{ReLU}} 32 \xrightarrow[5]{\text{convp}} \xrightarrow[\text{BN}]{\text{ReLU}} 128 \xrightarrow[5]{\text{convp}} \xrightarrow[\text{BN}]{\text{ReLU}} 256 \xrightarrow[5]{\text{convp}} \xrightarrow[\text{BN}]{\text{ReLU}} 128 \xrightarrow[5]{\text{convp}} \xrightarrow[\text{BN}]{\text{ReLU}} 32 \xrightarrow[5]{\text{convp}} \xrightarrow{\text{Tanh}} 3$
G_3	$9 \xrightarrow[5]{\text{convp}} \xrightarrow[\text{BN}]{\text{ReLU}} 32 \xrightarrow[5]{\text{convp}} \xrightarrow[\text{BN}]{\text{ReLU}} 128 \xrightarrow[5]{\text{convp}} \xrightarrow[\text{BN}]{\text{ReLU}} 256 \xrightarrow[5]{\text{convp}} \xrightarrow[\text{BN}]{\text{ReLU}} 256 \xrightarrow[5]{\text{convp}} \xrightarrow[\text{BN}]{\text{ReLU}} 128 \xrightarrow[5]{\text{convp}} \xrightarrow[\text{BN}]{\text{ReLU}} 32 \xrightarrow[5]{\text{convp}} \xrightarrow{\text{Tanh}} 3$
G_4	$9 \xrightarrow[5]{\text{convp}} \xrightarrow[\text{BN}]{\text{ReLU}} 32 \xrightarrow[5]{\text{convp}} \xrightarrow[\text{BN}]{\text{ReLU}} 128 \xrightarrow[5]{\text{convp}} \xrightarrow[\text{BN}]{\text{ReLU}} 256 \xrightarrow[5]{\text{convp}} \xrightarrow[\text{BN}]{\text{ReLU}} 512 \xrightarrow[5]{\text{convp}} \xrightarrow[\text{BN}]{\text{ReLU}} 256 \xrightarrow[5]{\text{convp}} \xrightarrow[\text{BN}]{\text{ReLU}} 128 \xrightarrow[5]{\text{convp}} \xrightarrow[\text{BN}]{\text{ReLU}} 32 \xrightarrow[5]{\text{convp}} \xrightarrow{\text{Tanh}} 3$
G_4	$9 \xrightarrow[5]{\text{convp}} \xrightarrow[\text{BN}]{\text{ReLU}} 32 \xrightarrow[5]{\text{convp}} \xrightarrow[\text{BN}]{\text{ReLU}} 128 \xrightarrow[5]{\text{convp}} \xrightarrow[\text{BN}]{\text{ReLU}} 256 \xrightarrow[5]{\text{convp}} \xrightarrow[\text{BN}]{\text{ReLU}} 512 \xrightarrow[5]{\text{convp}} \xrightarrow[\text{BN}]{\text{ReLU}} 512 \xrightarrow[5]{\text{convp}} \xrightarrow[\text{BN}]{\text{ReLU}} 256 \xrightarrow[5]{\text{convp}} \xrightarrow[\text{BN}]{\text{ReLU}} 128 \xrightarrow[5]{\text{convp}} \xrightarrow[\text{BN}]{\text{ReLU}} 32 \xrightarrow[5]{\text{convp}} \xrightarrow{\text{Tanh}} 3$

Discriminators	
D_1	$3{*}4 \xrightarrow[3]{\text{conv}} \xrightarrow{\text{ReLU}} 32{*}4 \xrightarrow{\text{concat}} \xrightarrow[3]{\text{conv}} \xrightarrow[\text{BN}]{\text{ReLU}} 128 \xrightarrow[3]{\text{conv}} \xrightarrow[\text{BN}]{\text{ReLU}} 128 \xrightarrow{\text{FC}} \xrightarrow{\sigma} 1$
D_2	$3{*}4 \xrightarrow[5/2]{\text{conv}} \xrightarrow{\text{ReLU}} 32{*}4 \xrightarrow{\text{concat}} \xrightarrow[5]{\text{conv}} \xrightarrow[\text{BN}]{\text{ReLU}} 128 \xrightarrow[5]{\text{conv}} \xrightarrow[\text{BN}]{\text{ReLU}} 128 \xrightarrow[5]{\text{conv}} \xrightarrow[\text{BN}]{\text{ReLU}} 256 \xrightarrow{\text{FC}} \xrightarrow{\sigma} 1$
D_3	$3{*}4 \xrightarrow[5/2]{\text{conv}} \xrightarrow{\text{ReLU}} 32{*}4 \xrightarrow{\text{concat}} \xrightarrow[5/2]{\text{conv}} \xrightarrow[\text{BN}]{\text{ReLU}} 128 \xrightarrow[5]{\text{conv}} \xrightarrow[\text{BN}]{\text{ReLU}} 128 \xrightarrow[5]{\text{conv}} \xrightarrow[\text{BN}]{\text{ReLU}} 256 \xrightarrow[5]{\text{conv}} \xrightarrow[\text{BN}]{\text{ReLU}} 256 \xrightarrow{\text{FC}} \xrightarrow{\sigma} 1$
D_4	$3{*}4 \xrightarrow[5/2]{\text{conv}} \xrightarrow{\text{ReLU}} 32{*}4 \xrightarrow{\text{concat}} \xrightarrow[5/2]{\text{conv}} \xrightarrow[\text{BN}]{\text{ReLU}} 128 \xrightarrow[5/2]{\text{conv}} \xrightarrow[\text{BN}]{\text{ReLU}} 256 \xrightarrow[5]{\text{conv}} \xrightarrow[\text{BN}]{\text{ReLU}} 256 \xrightarrow[5]{\text{conv}} \xrightarrow[\text{BN}]{\text{ReLU}} 512 \xrightarrow{\text{FC}} \xrightarrow{\sigma} 1$
D_5	$3{*}4 \xrightarrow[5/2]{\text{conv}} \xrightarrow{\text{ReLU}} 32{*}4 \xrightarrow{\text{concat}} \xrightarrow[5/2]{\text{conv}} \xrightarrow[\text{BN}]{\text{ReLU}} 128 \xrightarrow[5/2]{\text{conv}} \xrightarrow[\text{BN}]{\text{ReLU}} 256 \xrightarrow[5/2]{\text{conv}} \xrightarrow[\text{BN}]{\text{ReLU}} 512 \xrightarrow[5]{\text{conv}} \xrightarrow[\text{BN}]{\text{ReLU}} 512 \xrightarrow{\text{FC}} \xrightarrow{\sigma} 1$

Table 4.2: Frame prediction Model 2 architectures. The discriminators process each frame independently in the first layer, before concatenating the features planes (represented by the layer $\xrightarrow{\text{concat}}$).

4.3.1 Network architectures

The parameters of both models are shown in Table 4.1 and Table 4.1. A significant difference between the two models is that Model 2 is using Batch Normalization [Ioffe and Szegedy, 2015] after each convolution (except the last convolution of the generators and the first convolution of the discriminators), while Model 1 does not. As observed in other works (such as [Radford et al., 2015]), we observed that Batch Normalization works better when the minibatches are separated between real and fake samples, which may not be intuitive. Model 1 is trained with SGD using $p = 1$, $\lambda_p = 1$, $\lambda_{adv} = 0.05$, $\alpha = 1$, and minibatches on size 4 or 8. In order to evaluate the quality of the predictions, we compare different baseline models to Model 1 using standard evaluation methods. The parameters of the baseline models are displayed in Table 4.3. On the other hand, Model 2 is trained with Adam, minibatches of size 16, $\lambda_p = 0$ and $\lambda_{gdl} = 0$ (pure adversarial setup). Model 1 does not use noise ϵ (see section 4.4) while Model 2 has additive noise in the middle features of each generator G.

	N_{scales}	λ_{L^p}	p	λ_{adv}	λ_{gdl}	α
single sc. L^2	1	1	2	0	0	
L^2	4	1	2	0	0	
L^1	4	1	1	0	0	
GDL L^1	4	1	1	0	1	1
Adv	4	1	1	0.05	0	

Table 4.3: Parameters of the baseline models compared to Model 1.

Generative model training: The generative model G architecture is presented in Table 4.1 for Model 1, and Table 4.2 for Model 2. It contains padded convolutions interlaced with ReLU nonlinearities. A hyperbolic tangent (Tanh) is added at the end of the model to ensure that the output values are between -1 and 1. For Model 1, the learning rate ρ_G starts at 0.04 and is reduced over time to 0.005. We train the network on small patches, and since it is fully convolutional, we can seamlessly apply it on larger images at test time. For Model 2, the learning rate ρ_G is set to 0.0002 (the scale is different since it is trained with Adam).

Adversarial training: The discriminative model D, also presented in Table 4.1 and Table 4.2, uses standard non-padded convolutions followed by fully connected layers and ReLU nonlinearities. The network is trained by setting the learning rate ρ_D to 0.02. For Model 2, we set the learning rate to 0.0002.

4.3.2 Quantitative evaluations

To evaluate the quality of the image predictions resulting from the different tested systems, we compute the Peak Signal to Noise Ratio (PSNR) between the true frame y and the prediction $\hat{y}$:

$$\text{PSNR}(y, \hat{y}) = 10 \log_{10} \frac{\max_{\hat{y}}^2}{\frac{1}{N} \sum_{i=0}^{N} (y_i - \hat{y}_i)^2}, \tag{4.7}$$

where $\max_{\hat{y}}$ is the maximum possible value of the image intensities. We also provide the Structural Similarity Index Measure (SSIM) of [Wang et al., 2004]. It ranges between -1 and 1, a larger score meaning a greater similarity between the two images.

To measure the loss of sharpness between the true frame and the prediction, we define the following sharpness measure based on the difference of gradients between two images y and $\hat{y}$:

$$\text{Sharp. diff.}(y, \hat{y}) = 10 \log_{10} \frac{\max_{\hat{y}}^2}{\frac{1}{N}\left(\sum_i \sum_j |(\nabla_i y + \nabla_j y) - (\nabla_i \hat{y} + \nabla_j \hat{y})|\right)} \tag{4.8}$$

where $\nabla_i y = |y_{i,j} - y_{i-1,j}|$ and $\nabla_j y = |y_{i,j} - y_{i,j-1}|$.

As for the other measures, a larger score is better. These quantitative measures on 378 test videos from UCF101[1] are given in Table 4.4. As it is trivial to predict pixel values in static areas, especially on the UCF101 dataset where most of the images are still, we also show evaluation in the moving areas as displayed in Figure 4.4. To this end, we use the EpicFlow method of [Revaud et al., 2015], and compute the different quality measures only in the areas where the optical flow is higher than a fixed threshold[2].

The numbers clearly indicate that all strategies perform better than the L^2 predictions in terms of PSNR, SSIM and sharpness. The multi-scale model brings some improvement, but used with an L^2 norm, it does not outperform simple frame

[1] We extracted from the test set list video files every 10 videos, starting at 1, 11, 21 etc.

[2] We use default parameters for the Epic Flow computation, and transformed the `.flo` file to `.png` using the Matlab code `http://vision.middlebury.edu/flow/code/flow-code-matlab.zip`. If at least one color channel is lower than 0.2 (image color range between 0 and 1), we replace the corresponding pixel intensity of the output and ground truth to 0, and compute similarity measures in the resulting masked images.

	1st frame prediction scores			2nd frame prediction scores		
	Similarity		Sharpness	Similarity		Sharpness
	PSNR	SSIM		PSNR	SSIM	
single sc. L^2	19.0	0.59	17.8	14.2	0.48	17.5
L^2	20.1	0.64	17.8	14.1	0.50	17.4
L^1	22.3	0.74	18.5	16.0	0.56	17.6
GDL L^1	23.9	0.80	18.7	18.6	0.64	17.7
Adv*	24.4	0.77	18.7	18.9	0.59	17.3
Model 1* (Adv+GDL)	27.2	0.83	19.6	22.6	0.72	18.5
Model 1 fine-tuned*	29.6	**0.90**	20.3	**26.0**	0.83	19.4
Last input	**30.0**	**0.90**	**22.1**	25.8	**0.84**	**20.3**

(a) Evaluation on the full patch

	1st frame prediction scores			2nd frame prediction scores		
	Similarity		Sharpness	Similarity		Sharpness
	PSNR	SSIM		PSNR	SSIM	
single sc. L^2	26.5	0.84	24.7	22.4	0.82	24.2
L^2	27.6	0.86	24.7	22.5	0.81	24.2
L^1	28.7	0.88	24.8	23.8	0.83	24.3
GDL L^1	29.4	0.90	25.0	24.9	0.84	24.4
GDL L^{1*}	29.9	0.90	25.0	26.4	0.87	24.5
Adv*	30.6	0.89	25.2	26.1	0.85	24.2
Model 1* (Adv+GDL)	31.5	0.91	**25.4**	28.0	0.87	**25.1**
Model 1 fine-tuned*	**32.0**	0.92	**25.4**	**28.9**	0.89	25.0
Last input	28.6	0.89	24.6	26.3	0.87	24.2
Optical flow	31.6	**0.93**	25.3	28.2	**0.90**	24.7

(b) Evaluation only in the areas of movement

Table 4.4: Quantitative evaluation of frame predictions on 10% of the UCF101 test images. The different models have been trained given 4 frames to predict the next one. Our best model has been fine-tuned on UCF101 after the training on Sports1m. Models with a * are fine-tuned on patches of size 64×64.

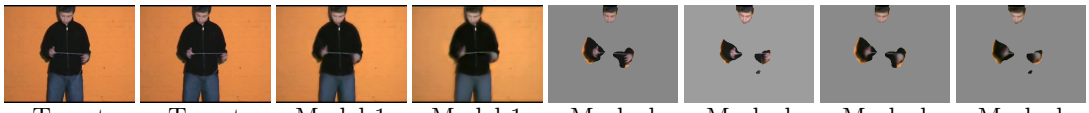


Figure 4.4: We report quantitate results computed on the whole image (Table 4.4a) and only on the areas of motion (Table 4.4b). The masks are based on optical flow intensity.

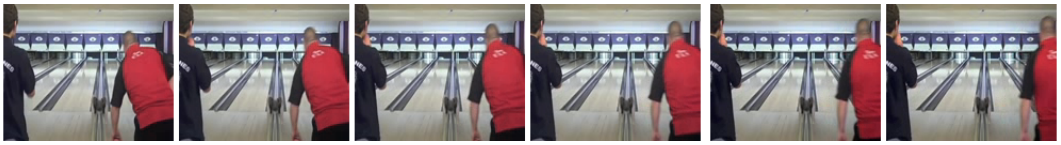
copy in the moving areas. The L^1 model improves the results, since it replaces the mean by the median value of individual pixel predictions. The GDL and adversarial predictions are leading to further gains, and finally the combination of the multi-scale, L^1 norm, GDL and adversarial training achieves the best PSNR, SSIM and Sharpness difference measure.

It is interesting to note that while we showed that the L^2 norm was a poor metric for training predictive models, the PSNR at test time is the worst for models trained optimizing the L^2 norm, although the PSNR is based on the L^2 metric. We postulate that the more robust loss is able to reach better minima, instead of stopping in a bad local minimum. We also include the baseline presented in [Ranzato et al., 2014] – courtesy of Piotr Dollar – that extrapolates the pixels of the next frame by propagating the optical flow from the previous ones.

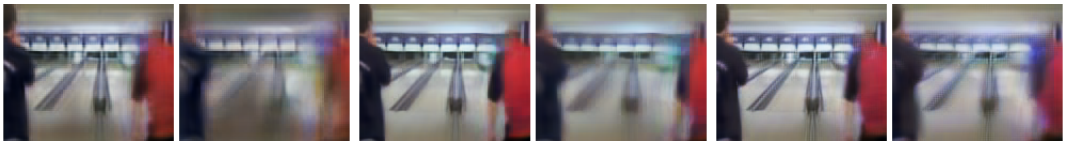
4.3.3 Frame prediction qualitative results

Figures 4.5 and 4.6 show results on test sequences from the Sport1m dataset. We predict two frames by using the model to predict the first frame, and we reuse this frame as an input, to predict a second frame. Although the model has not been trained in order to perform in this mode, if the first prediction is on the manifold of data, the second prediction should also be on the manifold. Figure 4.5 shows good generations, while Figure 4.6 shows a failure case. The failure is likely due to the large motion present in the input frames.

In order to evaluate our model on the UCF101 dataset, we first compare our results to Ranzato et al. [Ranzato et al., 2014]. To obtain grayscale images, we make RGB predictions and extract the Y channel of our Model 1. Images from [Ranzato et al., 2014] are generated by averaging 64 results obtained using different

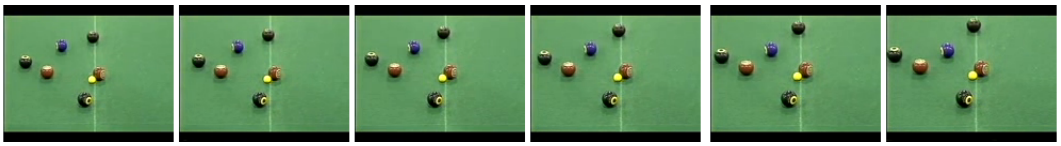


(a) Input frames (b) Groundtruth

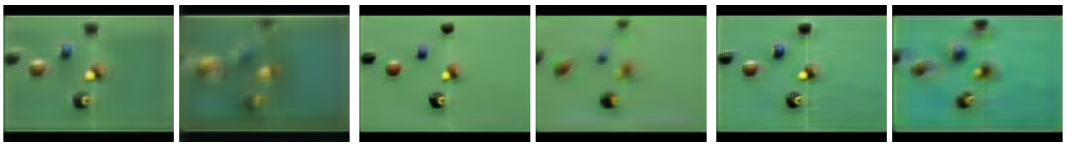


(c) L^2 predictions (d) L^1 predictions (e) GDL L^1 predictions

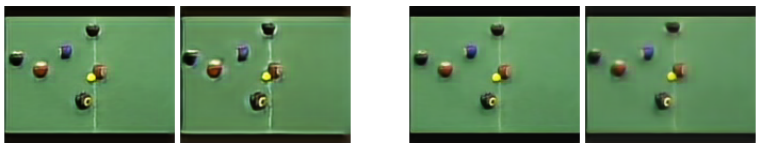
(f) Adversarial predictions (g) Adversarial+GDL predictions



(h) Input frames (i) Groundtruth

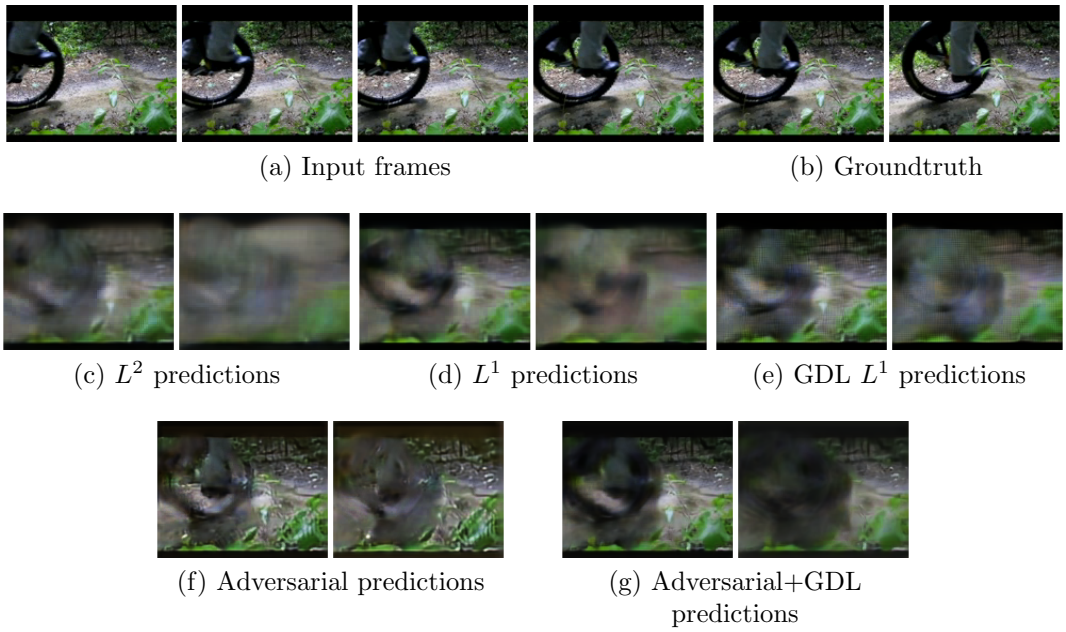


(j) L^2 predictions (k) L^1 predictions (l) GDL L^1 predictions



(m) Adversarial predictions (n) Adversarial+GDL predictions

Figure 4.5: Results on 2 video clips from Sport1m, using Model 1. The second prediction is obtained by applying the network recursively.



(a) Input frames (b) Groundtruth

(c) L^2 predictions (d) L^1 predictions (e) GDL L^1 predictions

(f) Adversarial predictions (g) Adversarial+GDL predictions

Figure 4.6: Predictions on a video clip from Sport1m, using Model 1. These predictions are a failure, which seems to be due to the aperture problem: despite the multi-scale approach, there is too much motion in the input.

tiling to avoid a blockiness effect, however creating instead a blurriness effect. The results can be seen in Figure 4.7.

We note that the results of Ranzato et al. appear slightly lighter than our results because of a normalization that does not take place in the original images, therefore the errors given here are not reflecting the full capacity of their approach. We tried to apply the blind deconvolution method of [Krishnan et al., 2011] to improve Ranzato et al. and our different results. As expected, the obtained sharpness scores are higher, but the image similarity measures are deteriorated because often the contours of the predictions do not match exactly the targets. More importantly, Ranzato et al. results appear to be more static in moving areas. Visually, the optical flow result appears similar to the target, but a closer look at thin details reveals that lines, heads of people are bent or squeezed.

(a) Target

(b) Prediction using a constant optical flow
PSNR=25.4 (18.9), SSIM = 0.88 (0.56)

(c) Ranzato et al.
PSNR = 16.3 (15.1), SSIM = 0.70 (0.55)

(d) Model 1 (Adv+GDL L^1)
PSNR = 26.7 (19.0), SSIM = 0.89 (0.59)

(e) Target

(f) Prediction using a constant optical flow
PSNR = 24.7 (20.6), SSIM = 0.84 (0.72)

(g) Ranzato et al.
PSNR = 20.1 (17.8), SSIM = 0.72 (0.65)

(h) Model 1 (Adv+GDL L^1) result
PSNR = 24.6 (20.5), SSIM = 0.81 (0.69)

Figure 4.7: Comparison of results on the Basketball Dunk and Ice Dancing clips from UCF101 appearing in [Ranzato et al., 2014]. We display 2 frame predictions for each method along with 2 zooms of each image. The PSNR and SSIM values are computed in the moving areas of the images (More than the 2/3 of the pixels in these examples). The values in parenthesis correspond to the second frame predictions measures.

We also show some video predictions on the original RGB version of UCF101 dataset. There predictions and are shown in Figure 4.8, and animations are available at `http://cs.nyu.edu/~mathieu/iclr2016.html`,

Finally, we show results on the NYUDepth dataset. These predictions are used using Model 2, which is trained in pure adversarial setup. These predictions stay sharper over time, which allows making longer term predictions by applying the model recursively. Figure 4.9 shows some predictions on 64×64 patches.

4.4 Discussion

We provided a benchmark of several strategies for next frame prediction, by evaluating the quality of the prediction in terms of Peak Signal to Noise Ratio, Structural Similarity Index Measure and image sharpness. These measure of quality, however, do not fully evaluate the capacity of the model to represent complex distributions. If the generator make a realistic prediction, but this prediction is different from the groundtruth, the PSNR and SSIM will not attribute a good score. Evaluation of generative models, and particularity GANs, which do not provide the possibility to evaluate the likelihood of the data under the model, is still an open research topic.

As mentioned in the previous sections, Model 1 does not use noise in the generator. This may seem like an odd choice, since the purpose of using GANs was to model multimodal distribution. However, although GANs should in theory generate points from multimodal distributions, in practice they often suffer from the Mode Collapse problem [Metz et al., 2016, Donahue et al., 2016]. Often, the generator generates samples from a single mode, and attributes very low proba-

(a) (b) (c)

Figure 4.8: Frame predictions on the UCF101 dataset, using Model 1. The images with a red border are predictions. Animated versions of the predictions (easier to interpret) can be found at `http://cs.nyu.edu/~mathieu/iclr2016.html`.

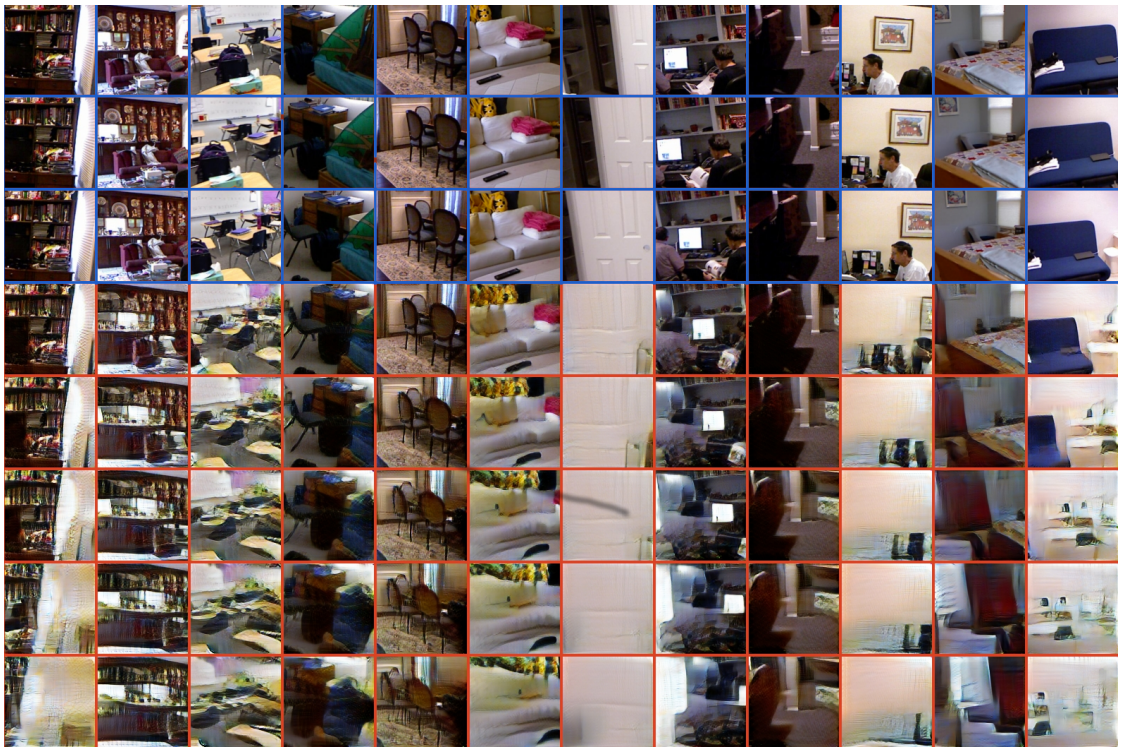


Figure 4.9: Frame predictions on the NYUDepth dataset, using Model 2, on 64×64 patches. The three first lines are the input while the rest are the predictions. We apply the model recursively to produce 5 images.

bilities to the other modes. This behavior can still be argued to be better than predicting the average, as would a pixel loss do, since there it carries more signal. It can be interpreted as performing a MAP inference of y rather than sampling. Therefore, rather than trying to address this problem, Model 1 is trained to focus on predicting the most likely output (MAP inference), by not using noise. Model 2 has additive noise in each scale, but varying the noise does not produce much variability in the predictions.

The presented architectures and losses may be used as building blocks for more sophisticated prediction models, involving memory and recurrence. Unlike most optical flow algorithms, the model is fully differentiable, so it can be fine-tuned for another task if necessary. Future work could deal with the evaluation of the classification performances of the learned representations in a weakly supervised context, for instance on the UCF101 dataset. Another extension of this work could be the combination of the current system with optical flow predictions. Alternatively, some applications, which currently use optical flow to perform another task, could benefit from directly predicting the next frame. A simple example is causal (where the next frame is unknown) segmentation of video streams.

Chapter 5

Disentangling factors of variation in deep representations using adversarial training

5.1 Introduction

A fundamental challenge in understanding sensory data is learning to disentangle the underlying factors of variation that give rise to the observations [Bengio, 2009]. For instance, the factors of variation involved in generating a speech recording include the speaker's attributes, such as gender, age, or accent, as well as the intonation and words being spoken. Similarly, the factors of variation underlying the image of an object include the object's physical representation and the viewing conditions. The difficulty of disentangling these hidden factors is that, in most real-world situations, each can influence the observation in a different and unpredictable way. It is seldom the case that one has access to rich forms of labeled

data in which the nature of these influences is given explicitly.

Often, the purpose for which a dataset is collected is to further progress in solving a certain supervised learning task. This type of learning is driven completely by the labels. The goal is for the learned representation to be invariant to factors of variation that are uninformative to the task at hand. While recent approaches for supervised learning have enjoyed tremendous success, their performance comes at the cost of discarding sources of variation that may be important for solving other, closely-related tasks. Ideally, we would like to be able to learn representations in which the uninformative factors of variation are separated from the informative ones, instead of being discarded. In particular, one factor of variation is time. Video prediction, as exposed in the previous chapters, can be understood as moving along one particular axis of variation, the temporal dimension.

Many other exciting applications require the use of generative models that are capable of synthesizing novel instances where certain key factors of variation are held fixed. Unlike classification, generative modeling requires preserving all factors of variation. But merely preserving these factors is not sufficient for many tasks of interest, making the disentanglement process necessary. For example, in speech synthesis, one may wish to transfer one person's dialog to another person's voice. Inverse problems in image processing, such as denoising and super-resolution, require generating images that are perceptually consistent with corrupted or incomplete observations.

In this chapter, we introduce a deep conditional generative model that learns to separate the factors of variation associated with the labels from the other sources of variability. We only make the weak assumption that we are able to distinguish between observations assigned to the same label during training. To make dis-

entanglement possible in this more general setting, we leverage both Variational Auto-Encoders (VAEs) [Kingma and Welling, 2013,Rezende et al., 2014] and Generative Adversarial Networks (GANs) [Goodfellow et al., 2014].

Our approach requires neither matching observations nor labels aside from the class identities. These properties allow the model to be trained on data with a large number of labels, enabling generalizing over the classes present in the training data.

5.2 Background

5.2.1 Variational autoencoder

The VAE framework is an approach for modeling a data distribution using a collection of independent latent variables. Let Y be a random variable (real or binary) representing the observed data and Z a collection of real-valued latent variables. The generative model over the pair (Y, Z) is given by $p_{\text{Dec}}(Y, Z) = p_{\text{Dec}}(Y|Z)p_Z(Z)$, where $p_Z(Z)$ is the prior distribution over the latent variables and $p_{\text{Dec}}(Y|Z)$ is the conditional likelihood function. Generally, we assume that the components of Z are independent Bernoulli or Gaussian random variables. The likelihood function is parameterized by a deep neural network referred to as the *decoder.*

A key aspect of VAEs is the use of a learned approximate inference procedure that is trained purely using gradient-based methods [Kingma and Welling, 2013, Rezende et al., 2014]. This is achieved by using a learned approximate posterior $p_{\text{Enc}}(Z|Y) = N(\mu, \sigma I)$ whose parameters (μ, σ) are given by another deep neural network referred to as the *encoder.* Thus, we have $z \sim p_{\text{Enc}}(Z|Y = y)$ and $\tilde{y} \sim p_{\text{Dec}}(Y|Z = z)$. The parameters of these networks are optimized by minimizing

the upper-bound on the expected negative log-likelihood of Y, which is given by

$$\mathbb{E}_{p_{\text{Enc}_\theta}(Z|Y=y)}[-\log p_{\text{Dec}_\theta}(Y|Z=z)] + D_{\text{KL}}(p_{\text{Enc}_\theta}(Z|Y=y)||p_Z(Z)). \tag{5.1}$$

The first term in Equation (5.1) corresponds to the reconstruction error, and the second term is a regularizer that ensures that the approximate posterior stays close to the prior. See section 2.1.3.1 for more details about Variational auto-encoders.

5.2.2 Generative adversarial networks

Generative Adversarial Networks (GANs) [Goodfellow et al., 2014] have enjoyed great success at producing realistic natural images [Radford et al., 2015]. See sections 2.1.3.2 and 4.2.2 for more details about GANs. The generator network, which task is to produce realistic samples, is denoted G_θ, while the discriminator, which is trained to differentiate between real and generated samples, is denoted D_ϕ. The entire trainable parameters of G_θ are stored in θ while the trainable parameters of D_ϕ are stored in ϕ. The goal is for the generator to produce increasingly more realistic images as the discriminator learns to pick up on increasingly more subtle inaccuracies that allow it to tell apart real images from fake ones, as the training procedure establishes a min-max game between the two networks.

Both D_ϕ and G_θ can be conditioned on the label of the input that we wish to classify or generate [Mirza and Osindero, 2014]. This approach has been successfully used to produce samples that belong to a specific class or possess some desirable property [Denton et al., 2015, Mathieu et al., 2015, Radford et al., 2015], as explained in the previous chapter for instance. As before, we define the objective of GANs as a stable point (or Nash equilibrium) in a two-player game, where

the generator G_θ aims at minimizing the loss $\mathcal{L}_G^{\text{adv}}$ and the discriminator D_ϕ the loss $\mathcal{L}_D^{\text{adv}}$:

$$\mathcal{L}_D^{\text{adv}}(\phi, x, y, \epsilon) = L^{bce}(D_\phi(x, y), 1) + L^{bce}(D_\phi(x, G_\theta(x, \epsilon)), 0) \tag{5.2}$$

$$\mathcal{L}_G^{\text{adv}}(\theta, x, \epsilon) = L^{bce}(D_\phi(x, G_\theta(x, \epsilon)), 1) \tag{5.3}$$

where x is a conditioning variable (in this chapter, it will be a class label), ϵ is a noise vector drawn from a prior distribution (e.g. $\mathcal{N}(0, I)$), and L^{bce} is the binary cross entropy loss, defined in section 4.2.2.

5.3 Model

5.3.1 Conditional generative model

We introduce a conditional probabilistic model admitting two independent sources of variation: an observed variable S that characterizes the specified factors of variation, and a continuous latent variable Z that characterizes the remaining variability. The variable S is a vector of real numbers, rather than a class ordinal or a one-hot vector, as we intend for the model to generalize to unseen identities.

Given an observed specified component s, we can sample

$$z \sim p_Z(Z) = \mathcal{N}(0, I) \quad \text{and} \quad y \sim p_{\text{Dec}_\theta}(Y|Z = z, S = s), \tag{5.4}$$

in order to generate a new instance y compatible with s.

The variables S and Z are marginally independent, which promotes disentanglement between the specified and unspecified factors of variation. Again here,

$p_{\mathrm{Dec}_\theta}(Y|Z=z,S=s)$ is a likelihood function described by a decoder network, Dec, and the approximate posterior $p(Z|Y)$ is modeled using a Gaussian distribution, $p_{\mathrm{Enc}_\theta}(Z|Y=y)=\mathcal{N}(\mu,\sigma I)$, whose parameters (μ,σ) are specified via an encoder network, Enc. In this new setting, the variational upper-bound is be given by

$$\mathbb{E}_{p_{\mathrm{Enc}_\theta}(Z|Y=y)}\left[-\log p_{\mathrm{Dec}_\theta}(Y=y|Z=z,S=s)\right]+D_{\mathrm{KL}}\left(p_{\mathrm{Enc}_\theta}(Z|Y=y)||p_Z(Z)\right). \tag{5.5}$$

The specified component s can be obtained from one or more images belonging to the same class. In this work, we consider the simplest case in which s is obtained from a single image. To this end, we define a deterministic encoder f_s that maps images to their corresponding specified components. All sources of stochasticity in S come from the data distribution. The conditional likelihood given by (5.4) can now be written as $\tilde{y}\sim p_{\mathrm{Dec}_\theta}(Y|Z=z,S=f_s(y))$ where y and $\tilde{y}$ share the same label. In addition to f_s, the model has an additional encoder f_z that parameterizes the approximate posterior $p_{\mathrm{Enc}}(Z|Y=y)$. It is natural to consider an architecture in which parameters of both encoders are shared.

We now define a single encoder Enc by

$$\mathrm{Enc}(y)=(f_s(y),f_z(y))=\big(s,(\mu,\sigma)\big)=(s,z), \tag{5.6}$$

where s is the specified component, and $z=(\mu,\sigma)$ the parameters of the approximate posterior that constitute the unspecified component. To generate a new instance, we synthesize s and z using Dec to obtain $\tilde{y}=\mathrm{Dec}(s,z)$.

The model described above cannot be trained by minimizing the log-likelihood alone. In particular, there is nothing that prevents all of the information about

the observation from flowing through the unspecified component, Z, which would result in training a conditional VAE [Sohn et al., 2015]. The decoder could learn to ignore S, and the approximate posterior could map images belonging to the same class to different regions of the latent space. This degenerate solution can be easily prevented when we have access to labels for the unspecified factors of variation, as in [Reed et al., 2015]. In this case, we could enforce that S is informative by requiring Dec to be able to reconstruct two observations having the same unspecified label after their unspecified components are swapped. But for many real-world scenarios, it is either impractical or impossible to obtain labels for the unspecified factors of variation. In the following section, we explain a way of eliminating the need for such labels.

5.3.2 Discriminative regularization

An alternative approach to preventing the degenerate solution described in the previous section, without the need for labels for the unspecified components, makes use of GANs. As before, we employ a procedure in which the unspecified components of a pair of observations are swapped. But since the observations need not be aligned along the unspecified factors of variation, it no longer makes sense to enforce reconstruction. After swapping, the class identities of both observations will remain the same, but the sources of variability within their corresponding classes will change. Hence, rather than enforcing reconstruction, we ensure that both observations are assigned high probabilities of belonging to their original classes by an external discriminator. Formally, we introduce the discriminative

term given by Equation (5.3) into the loss given by Equation (5.5), yielding

$$\mathbb{E}_{p_{\text{Enc}_\theta}(Z|Y=y)}\Big[-\log p_{\text{Dec}_\theta}(Y=y|Z=z,S=s)+\lambda\mathbb{E}_{p_{\text{Dec}_\theta}(Y|Z=z,S=s)}\left[-\log D_\phi(s,y)\right]\Big] \\ +D_{\text{KL}}(p_{\text{Enc}_\theta}(Z|Y=y)||p_Z(Z)) \tag{5.7}$$

where λ is a non-negative weight. The difference with Equation (5.5) is the second term in the expected value. It corresponds to the expected value of the generator loss $\mathcal{L}_G^{\text{adv}}$ of the adversarial setting in Equation (5.3). In practice, this expected value is approximated using the MAP, which corresponds to approximate the Gaussian by its mean.

Recent works have explored combining VAE with GAN [Larsen et al., 2015, Dumoulin et al., 2016]. These approaches aim at including a recognition network (allowing solving inference problems) to the GAN framework. In the setting used in this work, GANs are used to compensate the lack of aligned training data. The work in [Larsen et al., 2015] investigates the use of GANs for obtaining perceptually better loss functions (beyond pixels). While this is not the goal of our work, our framework is able to generate sharper images, which comes as a side effect. We evaluated including a GAN loss also for samples, however, the system became unstable without leading to perceptually better generations. An interesting variant could be to use separate discriminator for images generated with and without supervision.

5.3.3 Training procedure

Let y_1 and y_1' be samples sharing the same label, namely id_1, and y_2 a sample belonging to a different class, id_2. On one hand, we want to minimize the upper

bound of negative log likelihood of y_1 when feeding to the decoder inputs of the form $(z_1, f_s(y_1))$ and $(z_1, f_s(y_1'))$, where z_1 are samples from the approximate posterior $p_{\text{Enc}_\theta}(Z|Y = y_1)$. On the other hand, we want to minimize the adversarial loss of samples generated by feeding to the decoder inputs given by $(z_1, f_s(y_2))$, where z_1 is sampled from the approximate posterior $p_{\text{Enc}_\theta}(Z|Y = y_1)$. This corresponds to swapping specified and unspecified factors of y_1 and y_2. We could only use upper bound if we had access to aligned data. Additionally, in order to enforce reconstructions to be realistic when z is sampled from $\mathcal{N}(0, I)$, we also generate $y_{\cdot 2} = \text{Dec}(z, f_s(y_2))$ where $z \sim \mathcal{N}(0, I)$ and use the discriminator as a loss. As in the GAN setting described in section 5.2.2, we alternate this procedure with updates of the adversary network. The diagram of the network is shown in figure 5.1, and the described training procedure is summarized in Algorithm 5.1.

5.4 Experiments

Datasets. We evaluate our model on both synthetic and real datasets: Sprites dataset [Reed et al., 2015], MNIST [LeCun et al., 1998], NORB [LeCun et al., 2004] and the Extended-YaleB dataset [Georghiades et al., 2001]. We used Torch7 [Collobert et al., 2011] to conduct all experiments. The network architectures follow that of DCGAN [Radford et al., 2015] and are described in Table 5.1.

Evaluation. We propose two forms of evaluation to illustrate the behavior of the proposed framework, one qualitative and one quantitative.

Qualitative evaluation is obtained by visually examining the perceptual quality of single-image analogies and conditional images generation. In all the experiments images were randomly chosen from the test set, see specific details for each dataset.

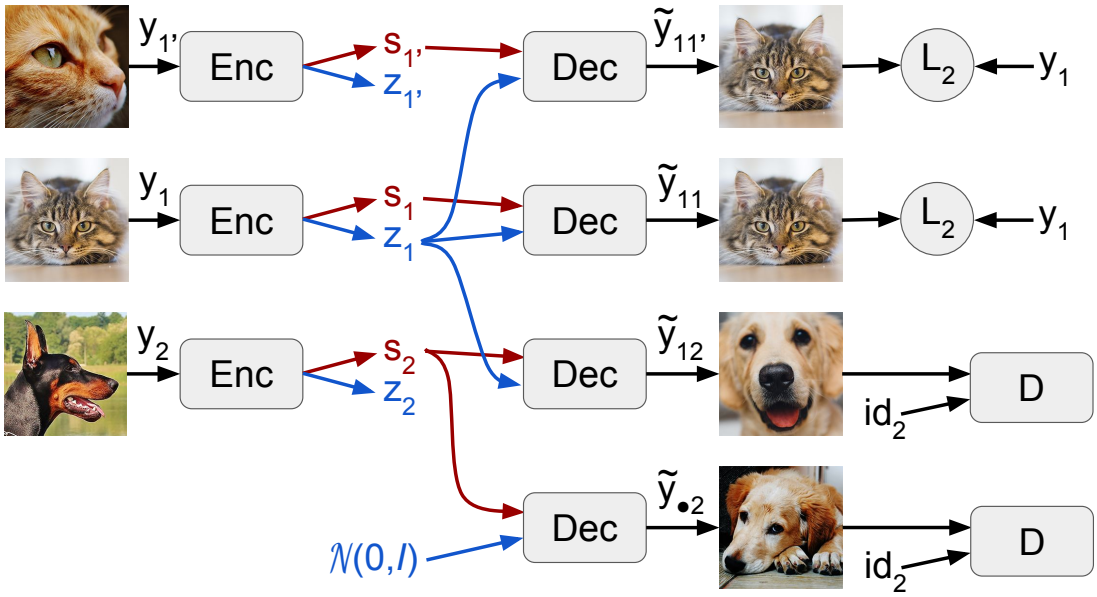


Figure 5.1: Training architecture. The inputs y_1 and y_1' are two different samples with the same label (here "cat"), whereas y_2 can have any label (for instance "dog"). The reconstructions $\tilde{y}_{11}$ and $\tilde{y}_{11'}$ are trained to match y_1, but y_{12} and $y_{\cdot 2}$ do not have an explicit target. We use the discriminator D to make them belong to class id_2 ("dog").

Algorithm 5.1 Full model training. The notations are defined in sections 5.2 and 5.3.

for number of training iterations **do**

Train the generative model

Sample datapoints y_1, y_1', y_2, where y_1 and y_1' have the same class label id_1

Compute $\begin{cases} (s_1, \mu_1, \sigma_1) = \text{Enc}_\theta(y_1) \\ (s_1', \mu_1', \sigma_1') = \text{Enc}_\theta(y_1') \\ (s_2, \mu_2, \sigma_2) = \text{Enc}_\theta(y_2) \end{cases}$ and sample $\begin{cases} z_1 \sim \mathcal{N}(\mu_1, \sigma_1) \\ z_1' \sim \mathcal{N}(\mu_1', \sigma_1') \\ z_2 \sim \mathcal{N}(\mu_2, \sigma_2) \end{cases}$

Compute the reconstructions $\tilde{y}_{11} = \text{Dec}_\theta(z_1, s_1), \quad \tilde{y}_{11'} = \text{Dec}_\theta(z_1, s_1')$

Back propagate the losses between $\tilde{y}_{11}$ and y_1, and between $\tilde{y}_{11'}$ and y_1

Compute $\tilde{y}_{12} = \text{Dec}_\theta(z_1, s_2)$

Backpropagate the adversarial loss $-\log(D_\phi(id_2, \tilde{y}_{12}))$, freezing the weights ϕ

Sample $z \sim \mathcal{N}(0, I)$, generate $\tilde{y}_{\cdot 2} = \text{Dec}_\theta(z, s_2)$

Backpropagate the adversarial loss $-\log(D_\phi(\tilde{y}_{\cdot 2}, id_2))$, freezing the weights ϕ

Train the discriminator

Sample datapoints y_1, y_2

Compute $\begin{cases} (s_1, \mu_1, \sigma_1) = \text{Enc}_\theta(y_1) \\ (s_2, \mu_2, \sigma_2) = \text{Enc}_\theta(y_2) \end{cases}$ and sample $\begin{cases} z_1 \sim \mathcal{N}(\mu_1, \sigma_1) \\ z_2 \sim \mathcal{N}(\mu_2, \sigma_2) \end{cases}$

Compute the reconstructions $\tilde{y}_{11} = \text{Dec}_\theta(z_1, s_1), \quad \tilde{y}_{21} = \text{Dec}_\theta(z_2, s_1)$

Backpropagate the adversarial loss $-\log(1 - D_\phi(\tilde{y}_{21}, id_1)) - \log(D_\phi(y_1, id_1))$ while keeping the weights θ frozen

end for

For all datasets, we evaluated the models in four different settings:

- *swapping:* given a pair of images, we generate samples by conditioning on the specified component extracted from one of the images, and sampling from the approximate posterior obtained from the other image. This procedure is analogous to the sampling technique employed during training, described in section 5.3.3, and corresponds to solving single-image analogies;

- *retrieval:* in order to assess the correlation between the specified and unspecified components, we performed nearest neighbor retrieval in the learned embedding spaces. We computed the corresponding representations for all samples (for the unspecified component we used the mean of the approximate posterior distribution) and then retrieved the nearest neighbors for a given query image;

- *interpolation:* to evaluate the coverage of the data manifold, we generated a sequence of images by linearly interpolating the codes of two given test images (for both specified and unspecified representations);

- *conditional generation:* given a test image, we generate samples conditioning on its specified component, sampling directly from the prior distribution, $p_Z(Z)$.

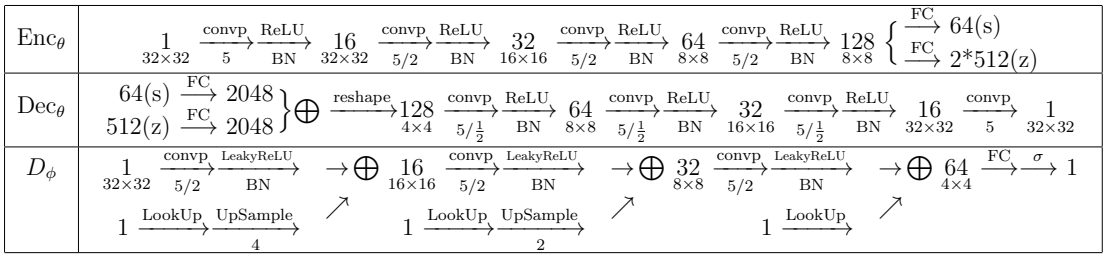
The objective evaluation of generative models is a difficult task and itself subject of current research [Theis et al., 2015]. Frequent evaluation metrics, such as measuring the log-likelihood of a set of validation samples, are often not very meaningful as they do not correlate to the perceptual quality of the images [Theis et al., 2015]. Furthermore, the loss function used by our model does not correspond a

bound on the likelihood of a generative model, which would render this evaluation less meaningful. As a quantitative measure, we evaluate the degree of disentanglement via a classification task. Namely, we measure how much information about the identity is contained in the specified and unspecified components. We also evaluate, on the Sprite dataset, the L^2 reconstruction loss for the swapping task. This is only possible since the Sprite dataset is a synthetic dataset with aligned data.

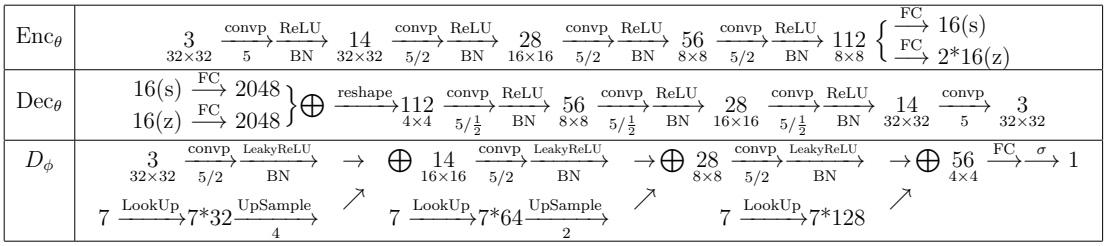
5.4.1 Network architectures

Model architectures are detailed in Table 5.1. The encoder consists of a shared sub-network that splits into two separate branches. In our experiments with MNIST and the Sprites datasets, the shared sub-network is composed by three 5x5 convolutional layers with stride 2, using spatial batch normalization (BN) [Ioffe and Szegedy, 2015] and ReLU non-linearities. For the NORB and YaleB datasets, we use six 3x3 convolutional layers, with stride 2 every other layer.

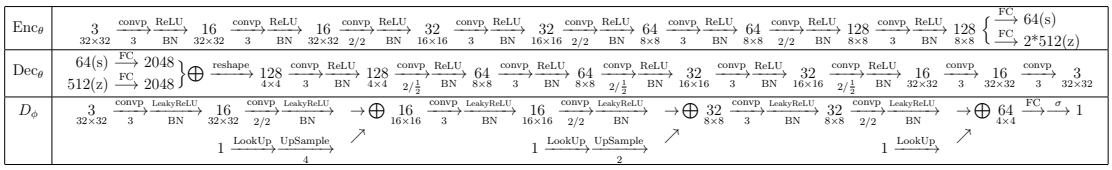
The output from the top convolution layer is split into two sub-networks. One parametrizes the approximate posterior of the unspecified component and consists of a fully-connected (FC) layer, producing two outputs corresponding to mean and variance of the approximate posterior (modeling the unspecified component). The other sub-network is also a fully connected used to produce the s vector modeling the specified component. The decoder network takes a sample z and a vector s as inputs. Both codes go through a fully connected network. These representations are merged together by directly adding them and fed into a feed-forward network composed by a network mirroring encoder structure (replacing the strides by fractional strides).



(a) MNIST Network



(b) Sprites Network



(c) NORB and YaleB networks

Table 5.1: Networks used for disentanglement. The decoders are conditioned on the identity S, which is going through three lookup tables. For sprites, the code S has 7 coordinates, which goes into 7 independent lookup tables. This is why the number of feature planes is a multiple of 7.

The discriminator is conditioned on the label, id, and configured following that used in (conditional) DCGAN. It contains three convolutional layers with stride 2, using batch normalization and Leaky-ReLU with slope 0.2. The label goes through three independent lookup tables and is added at the three first layers of representation.

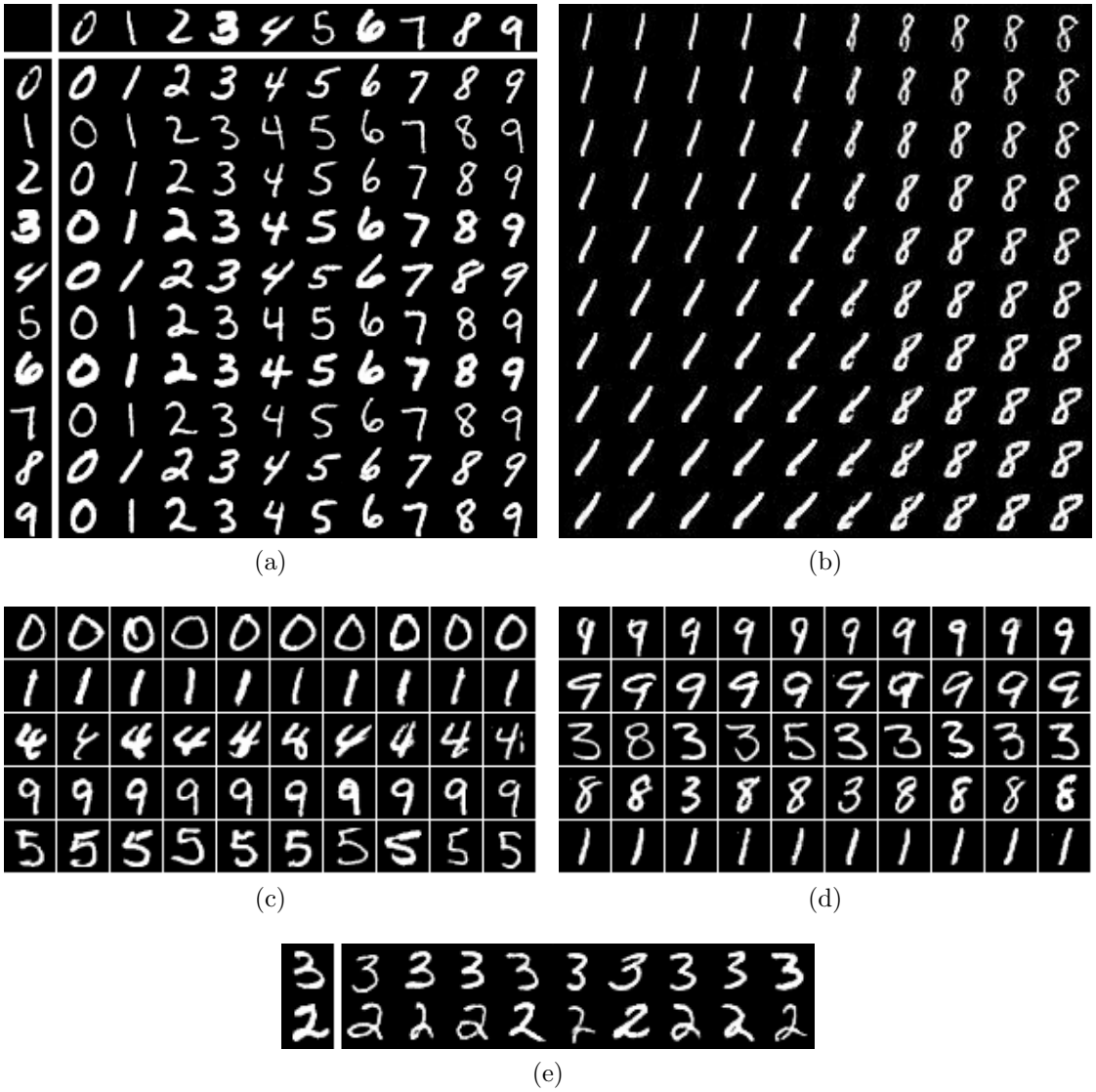
The dimensionality of each representation varies from dataset to dataset. They were obtained by monitoring the results on a validation set. For MNIST, we used 16 coefficients for each component. For sprites, NORB and Extended-YaleB, we set their dimensions as 64 and 512 for specified and unspecified components respectively. We found that using Stochastic Gradient Descent (SGD) gives good results.

5.4.2 MNIST

In this setup, the specified part is simply the class of the digit. The goal is to show that the model is able to learn to disentangle the style from the identity of the digit and to produce satisfactory analogies.

We cannot test the ability of the model to generalize to unseen identities. In this case, one could directly condition on a class label [Kingma et al., 2014, Makhzani et al., 2015]. It is still interesting that the proposed model is able to transfer handwriting style without having access to matched examples while still be able to learn a smooth representation of the digits as shown in the interpolation results.

Results for all four qualitative experiments are shown in Figure 5.2. We can see that both swapping and interpolation give very good results, showing that the model clearly learns to transfer handwritten styles from one digit to another.



(a) (b) (c) (d) (e)

Figure 5.2: (a): A visualization grid of 2D MNIST image swapping generation. The top row and leftmost column come from the test set. The other digits are generated using z from the leftmost digit, and s from the digit at the top of the column. The diagonal digits show reconstructions. (b): Interpolation visualization. Digits located at the top-left and bottom-right corners come from the dataset. The other digits are generated by interpolating s and z. Like (a), each row has constant a z each column a constant s. (c): Nearest neighbor retrieval querying on *specified* component s. Digits on the left are used as the query. (d): Nearest neighbor retrieval querying on *unspecified* component z. (e): Digit generation by sampling z and extracting s from the digit on the left.

5.4.3 Sprites dataset

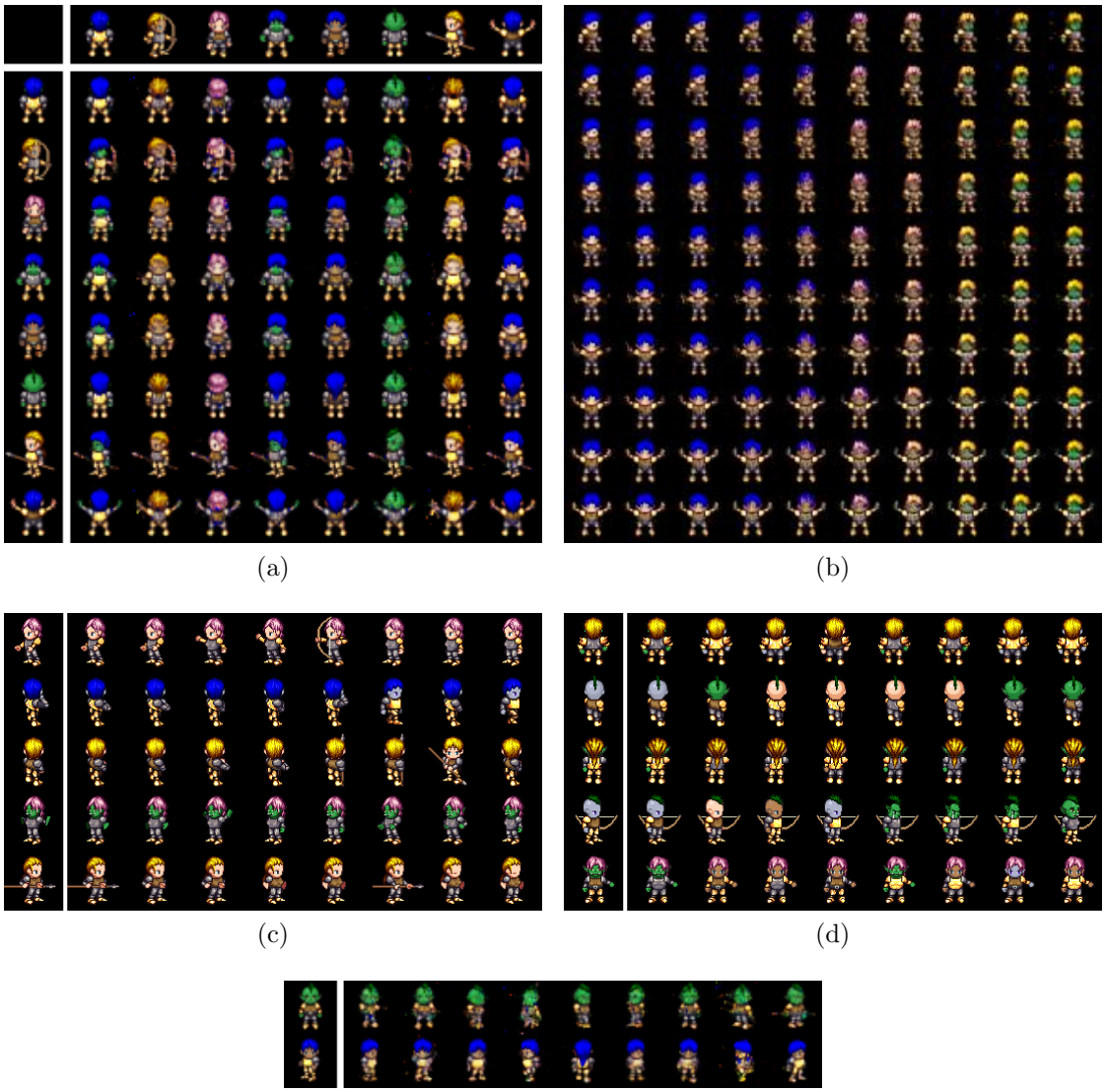
The Sprite dataset is composed of 672 unique manga-like characters (we refer to them as sprites), each of which is associated with 20 animations [Reed et al., 2015]. The automatic generation of sprites is motivated to reduce human effort in generating new drawings for animated movies and/or video games. Any sprite can present 7 sources of variation: body type, gender, hair type, armor type, arm type, greaves type, and weapon type, in addition to its pose.

Unlike the work in [Reed et al., 2015], we do not use any supervision regarding the positions of the sprites. It is important to point out that although the 2D sprites dataset was constructed in an aligned way, our setting doesn't make use the alignment. In this setting, the specified part of the sprites in the test set have not been seen during training, which shows the ability of our model to generalize on the s part as well as z. As in the previous section, the results obtained for all four qualitative experiments are shown in Figure 5.3.

The interpolation results show that one can smoothly transition between identities or positions. It is worth noting that this dataset has a fixed number of discrete positions. Thus, Figure 5.3b shows a reasonable coverage of the manifold with some abrupt changes. For instance, the hands are not moving up from the pixel space, but appearing gradually from the faint background.

5.4.4 NORB

For the NORB dataset we used instance identity (rather than object category) for defining the labels. This results in 25 different object identities in the training set and another 25 distinct objects identities in the testing set. As in the Sprite



(a) (b) (c) (d) (e)

Figure 5.3: (a): A visualization grid of 2D Sprites swapping generation. The top row and leftmost column come from the test set. The other sprites are generated using z from leftmost sprites, and s from the image at the top of the column. The diagonal images show reconstructions. (b): Interpolation visualization. Sprites located at the top-left and bottom-right corners come from the dataset. The other images are generated by interpolating s and z. Like (a), each row has constant a z each column a constant s. (c): Nearest neighbor retrieval querying on *specified* component s. Sprites on the left are used as the query. (d): Nearest neighbor retrieval querying on *unspecified* component z. (e): Sprites generation by sampling z and extracting s from the sprite on the left.

dataset, the identities used at testing have never been presented to the network at training time.

In this case, however, the small number of identities seen at training time makes the generalization more difficult. In Figures 5.4a and 5.4b, we present results for interpolation and swapping. We observe that the model is able to resolve analogies well. However, the quality of the results is degraded. In particular, classes having high variability (such as planes) are not reconstructed well. Also, some of the models are highly symmetric, thus creating a lot of uncertainty. We conjecture that these problems could be eliminated in the presence of more training data. Queries in the case of NORB are not as expressive as with the sprites, but we can still observe good behavior. They are shown in Figure 5.4c and 5.4d. Finally, Figure 5.4e shows generation with sampled z.

5.4.5 Extended-YaleB

The YaleB [Georghiades et al., 2001] dataset consists of facial images of 28 individuals taken under different positions and illuminations. The training and testing sets contains roughly 600 and 180 images per individual respectively. Figure 5.5 shows results for swapping, interpolation and sampling on a set of testing images. Due to the small number of identities, we cannot test in this case the generalization to unseen identities. We observe that the model is able to resolve the analogies in a satisfactory manner, position and illumination are transferred correctly although these positions have not been seen at train time for these individuals.

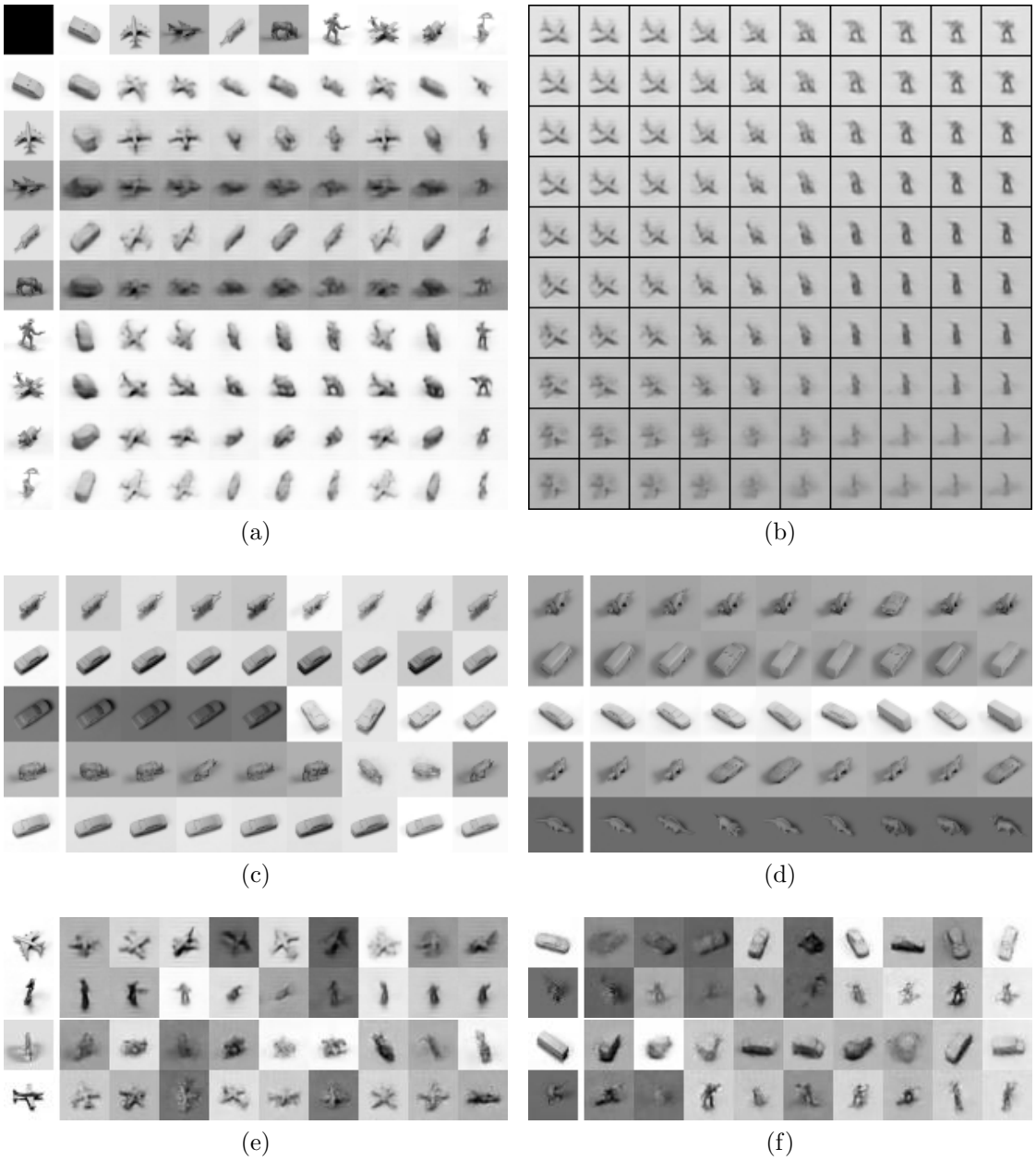


Figure 5.4: (a): A visualization grid of 2D NORB swapping generation. Top row and leftmost column come from the test set. Other images are generated using z from leftmost image, and s from top image. (b): Interpolation visualization. Images located at the top-left and bottom-right corners come from the dataset. The other images are generated by interpolating s and z. Like (a), each row has constant a z each column a constant s. (c): Nearest neighbor retrieval querying on *specified* component s. Images on the left are used as the query. (d): Nearest neighbor retrieval querying on *unspecified* component z. (e) and (f): Images generation by sampling z and extracting s from the image on the left.

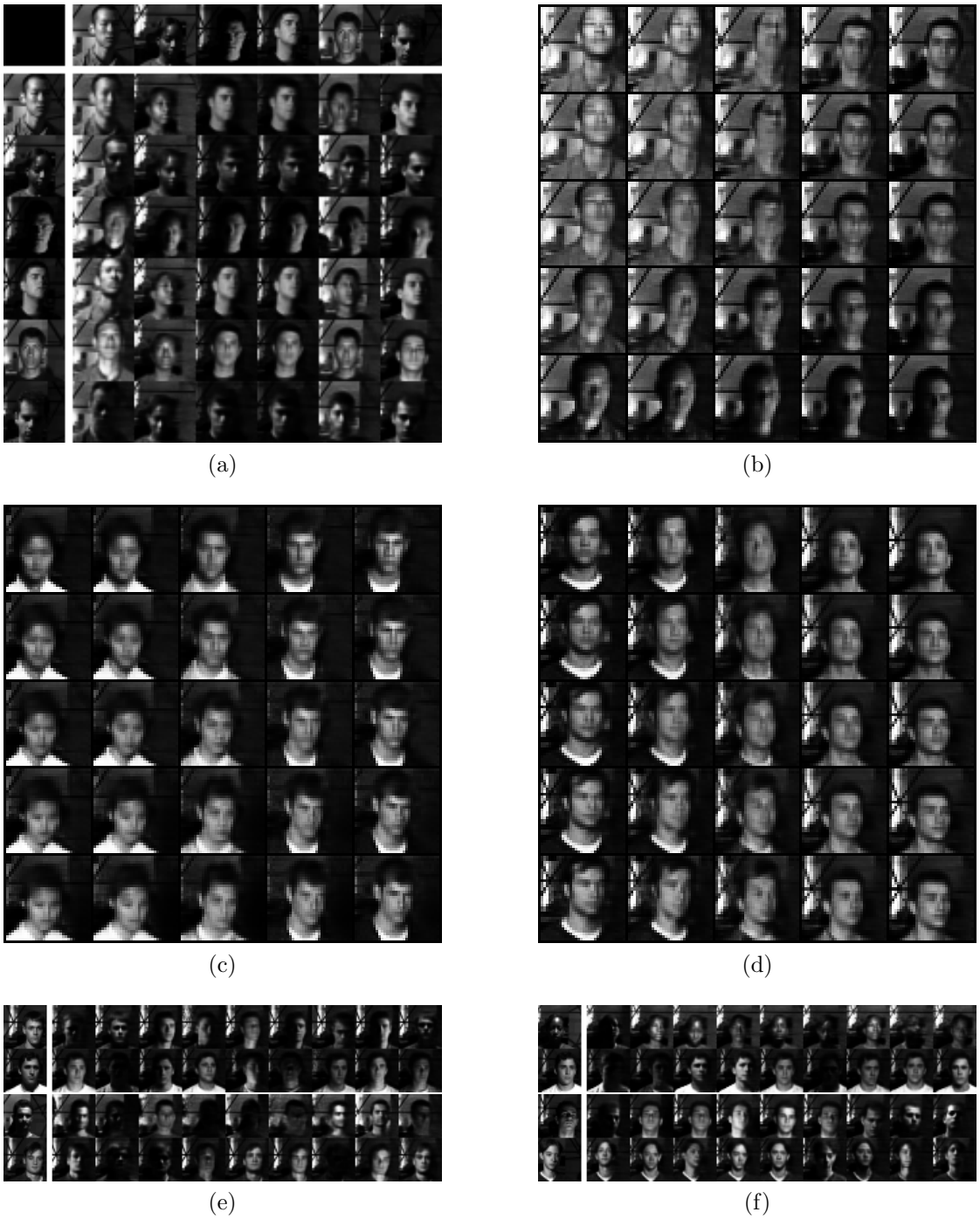


Figure 5.5: YaleB-Extended dataset. (a): swapping grid generation. Faces are generated using using z from leftmost image and s from top image. (b), (c) and (d): Interpolation visualization. Top-left and bottom-right corners from the test set, other faces generated by interpolating s on rows and z on columns. (e) and (f): Images generation by sampling z and extracting s from the image on the left.

5.4.6 Quantitative evaluation

Independence of s and z. We analyze the disentanglement of the specified and unspecified representations, by using them as input features for a prediction task. We trained a two-layer neural network with 256 hidden units to predict structured labels for the sprite dataset, toy category for the NORB dataset (four-legged animals, human figures, airplanes, trucks, and cars) and the subject identity for Extended-YaleB dataset. We used early-stopping on a validation set to prevent overfitting. We report both training and testing errors in Table 5.2. We also show the "chance" baseline, which corresponds to training a classifier with no input, i.e. learning the mean of the signal.

In all cases the unspecified component is agnostic to the identity information, almost as bad as the "chance" baseline. On the other hand, the specified components are highly informative, producing almost the same results as a classifier directly trained in a discriminative manner. In particular, we observe some overfitting in the NORB dataset. This might also be due to the difficulty of generalizing to unseen identities using a small dataset.

Table 5.2: Error rates for classification using z and s.

Data Set	**Sprites**		**NORB**		**Extended-YaleB**	
	z	s	z	s	z	s
train	58.6%	5.5%	79.8%	2.6%	96.4%	0.05%
test	59.8%	5.2%	79.9%	13.5%	96.4%	0.08%
no input (chance)	60%		80%		96.4%	

Influence of components of the framework. In order to access the advantage of jointly training the system to learn the specified and unspecified parts, we tried another training scheme, summarized in the following two-step approach:

- Add a two-layer neural network on top of the specified part of the encoder,

followed by a classification loss. Train this system in a plain supervised fashion to learn the class of the samples. When the system is converged, freeze the weights.

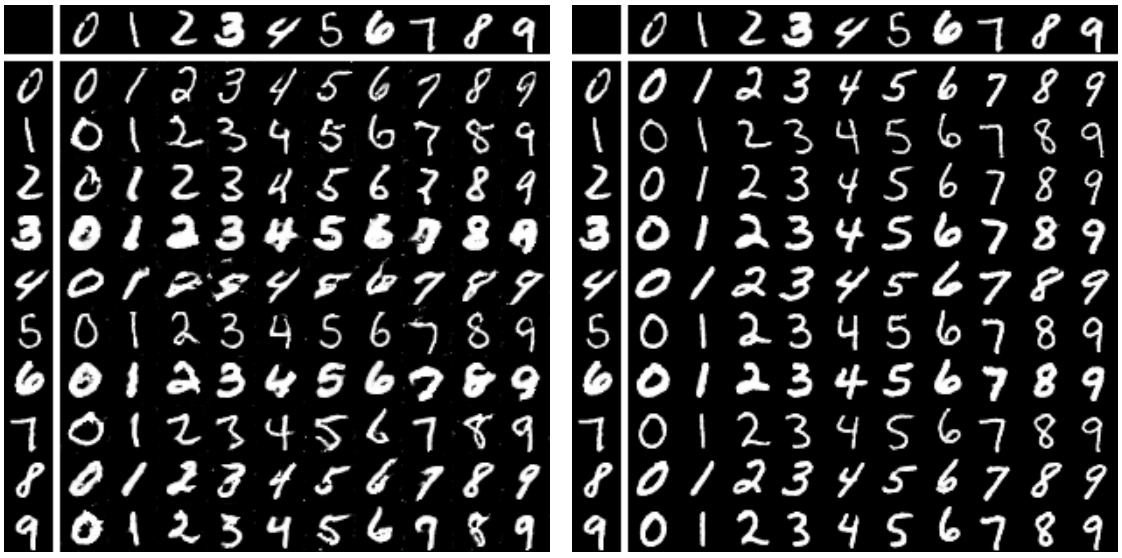
- Add another encoder to produce the unspecified part of the code, and train the system as before (keeping the weights of the "specified" encoder frozen).

We then evaluate the L^2 reconstruction loss on the Sprites dataset for the swapping experiment. Since we have perfect data alignment, we have access to the groundtruth and therefore we can compute the loss, even if the two inputs do not come from the same class. Note that this is not possible in most non-synthetic datasets. While the jointly trained approach reaches a L^2 loss of 0.014, a model that pretrains f_s for classification reaches an error of 0.024 (with almost twice as many parameters for the encoder, since the parameters are not shared anymore). A comparison of the generations with and without pretraining, on the MNIST dataset, is shown in Figure 5.6.

Finally, we assess the contribution of the conditional adversarial discriminator. In the model with a pretrained encoder for s, we disable the adversarial loss and train it normally (keeping the weights of the "specified" encoder frozen). This corresponds to training a Variational Auto-Encoder, conditioned on the pretrained s. In this case, the error goes up to 0.030, which shows that the discriminator is indeed useful.

5.5 Conclusions and discussion

This chapter presents a conditional generative model that learns to disentangle the factors of variations of the data specified and unspecified through a given cate-



(a) (b)

Figure 5.6: (a): A visualization grid MNIST swapping generation, where the encoder for the specified part of the code (s) has been pretrained for classification. (b): A visualization grid MNIST swapping generation, where the encoder is trained to jointly generate s and z. The generated digits are visually more satisfying.

gorization. The proposed model does not rely on strong supervision regarding the sources of variations. This is achieved by combining two very successful generative models: VAE and GAN. The model is able to resolve the analogies in a consistent way on several datasets with minimal parameter/architecture tuning.

Although these initial results are promising there is a lot to be tested and understood. The model is motivated on a general setting that is expected to be encountered in more realistic scenarios. However, in this initial study, we only tested the model on rather constrained examples. As was observed in the results shown using the NORB dataset, given the weaker supervision assumed in our setting, the proposed approach seems to have a high sample complexity relying on training samples covering the full range of variations for both specified and unspecified variations.

The proposed model does not attempt to disentangle variations within the specified and unspecified components. There are many possible ways of mapping a unit Gaussian to corresponding images, in the current setting, there is nothing preventing the obtained mapping to present highly entangled factors of variations.

Chapter 6

Energy-Based Generative Adversarial Networks

6.1 Introduction

In this chapter, we propose a novel approach to Generative Adversarial Networks (GANs). We view the discriminator D as an energy model, which aims at attributing an energy to each sample without explicit probabilistic interpretation. We refer to section 2.1.3.2 for details about GANs and section 2.1.2 for background on energy models. The energy function computed by the discriminator can be viewed as a trainable cost function for the generator. The discriminator is trained to assign low energy values to the regions of high data density, and higher energy values to the regions of low data density. Conversely, the generator can be viewed as a trainable parameterized function that produces samples in regions of the space to which the generator assigns low energy. While it is often possible to convert energies into probabilities through a Gibbs distribution [Lecun et al.,

2006], the absence of normalization in this energy-based form of GAN provides greater flexibility in the choice of architecture for the discriminator and in the training procedure.

The probabilistic binary discriminator from the original formulation of GANs can be seen as one way among many ways to define the contrast function and loss functional, as described in [Lecun et al., 2006] for the supervised and weakly supervised settings, and [Ranzato et al., 2007a] for the unsupervised setting. We experimentally demonstrate this concept, in the setting where the discriminator is an auto-encoder architecture, and the energy is the reconstruction error.

This chapter primarily casts GANs into an energy-based model scope. Consequently, the discriminator is seen as an energy model and can, in theory, be any of the previously developed energy models. Although this chapter presents implementation for a single approach, auto-encoders, and in this context can be viewed as a regularized auto-encoder (see section 6.2.3.1), other architectures for the discriminator could be used. Moreover, the task of generator is simply to produce contrastive samples, and could be replaced or extended by any approach producing contrastive samples, such as the use of noisy samples [Vincent et al., 2010] and noisy gradient descent methods including contrastive divergence [Carreira-Perpinan and Hinton, 2005].

The works of [Kim and Bengio, 2016] and [Nowozin et al., 2016] are closely related to this chapter, but differ on several points. In [Kim and Bengio, 2016], the authors propose a probabilistic GAN and cast it into an energy-based density estimator by using the Gibbs distribution. Unlike EBGAN, the proposed framework keeps working with probabilities, which results in a loss function that is very similar to the original GAN, and requires an ad hoc approximation of the entropy.

In [Nowozin et al., 2016], as in this chapter, the family of loss functions is extended. However, the class of functions covered is entirely different.

6.2 The EBGAN Model

Let p_{data} be the underlying probability density of the distribution that produces the dataset. The generator G is trained to produce a sample $G(\epsilon)$, for instance an image, from a random vector ϵ, which is sampled from a known distribution p_ϵ, for instance $\mathcal{N}(0, I)$. The discriminator D takes either real or generated inputs, and estimates the energy value $E \in \mathbb{R}$ accordingly, as explained later. For simplicity, we assume that D produces non-negative values, but the analysis would hold as long as the values are bounded below.

6.2.1 Objective functional

The output of the discriminator goes through an objective functional in order to shape the energy function to attribute low energy to the real data samples, and higher energy to the generated ("fake") ones. In this work, we use a margin loss, but many other choices are possible as detailed in [Lecun et al., 2006]. Similarly to what has been done with the classical probabilistic GAN [Goodfellow et al., 2014], in order to get better quality gradients when the generator is far from convergence, we use two different losses, one to train D and the other to train G.

For the sake of simplicity, this chapter considers unconditioned GANs, but this is not a restriction. Let m be a positive margin and f be a differentiable, strictly increasing, convex function such that $f(0) = 0$. Given a data sample x and a generated sample $G(\epsilon)$, the discriminator loss $\mathcal{L}_D$ and the generator loss $\mathcal{L}_G$ are

formally defined:

$$\mathcal{L}_D(x, \epsilon) = f\left(D(x)\right) + f\left(\left[m - D\left(G(\epsilon)\right)\right]^+\right) \tag{6.1}$$

$$\mathcal{L}_G(\epsilon) = f\left(D\left(G(\epsilon)\right)\right) \tag{6.2}$$

where $[\cdot]^+ = max(0, \cdot)$. Minimizing $\mathcal{L}_G$ with respect to the parameters of G is similar to maximizing the second term of $\mathcal{L}_D$: it has the same minimum but non-zero gradients when $D\left(G(\epsilon)\right) \geq m$.

6.2.2 Optimality of the solution

In this section, we present a theoretical analysis of the system presented in section 6.2.1. We show that if the system reaches a Nash equilibrium, then the generator G produces samples that are indistinguishable from the distribution of the dataset. However, it should be noted that this section is done in a non-parametric setting, i.e. we assume that D and G have infinite capacity. We also assume that p_{data} and p_G (defined later) have a density, which can always be made possible by adding a small amount of noise.

Given a generator G, let p_G be the density distribution of $G(\epsilon)$ where $\epsilon \sim p_\epsilon$. In other words, p_G is the density distribution of the samples generated by G.
We define

$$V(G, D) = \int_{x,\epsilon} \mathcal{L}_D(x, \epsilon) p_{\text{data}}(x) p_\epsilon(\epsilon) \mathrm{d}x \mathrm{d}\epsilon \tag{6.3}$$

$$U(G, D) = \int_{\epsilon} \mathcal{L}_G(\epsilon) p_\epsilon(\epsilon) \mathrm{d}\epsilon \tag{6.4}$$

We train the discriminator D to minimize the quantity V and the generator G to

minimize the quantity U.

A Nash equilibrium of the system is a pair (G^*, D^*) that satisfies:

$$V(G^*, D^*) \leq V(G^*, D) \qquad \forall D \tag{6.5}$$

$$U(G^*, D^*) \leq U(G, D^*) \qquad \forall G \tag{6.6}$$

We observe that

$$V(G, D) = \int_x \big(p_{\text{data}}(x) f(D(x)) + p_G(x) f([m - D(x)]^+)\big) \mathrm{d}x \tag{6.7}$$

$$U(G, D) = \int_x f(D(x)) p_G(x) \mathrm{d}x \tag{6.8}$$

Definition 6.1. *For any discriminator D, we define its clamped version, denoted $\widehat{D}$, defined by $\widehat{D}(x) = \min(m, D(x))$.*

Lemma 6.1. *For any generator G and any disriminator D, $V(G, \widehat{D}) \leq V(G, D)$.*

Proof.

$$V(G, \widehat{D}) = \int_x p_{\text{data}}(x) f\left(\widehat{D}(x)\right) + p_G(x) f\left(\left[m - \widehat{D}(x)\right]^+\right) \mathrm{d}x \tag{6.9}$$

$$= \int_x p_{\text{data}}(x) f\left(\widehat{D}(x)\right) + p_G(x) f\left([m - D(x)]^+\right) \mathrm{d}x \tag{6.10}$$

$$\leq \int_x p_{\text{data}}(x) f\left(D(x)\right) + p_G(x) f\left([m - D(x)]^+\right) \mathrm{d}x \tag{6.11}$$

since $\widehat{D}(x) \leq D(x)$ for all x and f is increasing. □

Besides, since D is continuous (it is represented by a neural network), Equation (6.11) is a strict inequality if there exists a non-zero measure set A such that for all x is A, $D(x) > m$ and $p_{\text{data}}(x) \neq 0$. Therefore any D^* that is part of a Nash

equilibrium satisfies $D^*(x) \in [0, m]$ for almost all x such that $p_{\text{data}}(x) \neq 0$.

Theorem 6.1. *If (D^*, G^*) is a Nash equilibrium of the system, then $p_{G^*} = p_{data}$ (almost everywhere), and $V(D^*, G^*) = 2f(m/2)$.*

Proof.

We show that $V(G^*, D^*) \leq 2f(m/2)$.

Let $D_{m/2}$ be the discriminator that produces the constant value $m/2$ (i.e. $\forall x, D_{m/2}(x) = m/2$). Using Equation (6.5), we obtain:

$$\begin{aligned} V(G^*, D^*) &\leq V(G^*, D_{m/2}) && (6.12) \\ &= \int_x f(m/2) p_{\text{data}}(x) \mathrm{d}x + \int_x f(m/2) p_{G^*}(x) \mathrm{d}x && (6.13) \\ &= 2f(m/2). && (6.14) \end{aligned}$$

We show that $V(G^*, D^*) \geq 2f(m/2)$.

Using equation (6.6), by using the ideal generator that produces p_{data} in place of G, we obtain:

$$\int_x f(D^*(x)) p_{G^*}(x) \mathrm{d}x \leq \int_x f(D^*(x)) p_{\text{data}}(x) \mathrm{d}x \tag{6.15}$$

so, adding the same quantity on both sides,

$$\int_x f(D^*(x)) p_{G^*}(x) \mathrm{d}x + \int_x f([m - D^*(x)]^+) p_{G^*}(x) \mathrm{d}x \leq V(G^*, D^*) \tag{6.16}$$

$$\int_x (f(D^*(x)) + f([m - D^*(x)]^+)) p_{G^*}(x) \mathrm{d}x \leq V(G^*, D^*) \tag{6.17}$$

and using the convexity of f, we get that $\forall x, f\left(\frac{D^*(x) + [m - D^*(x)]^+}{2}\right) \leq \frac{f(D^*(x)) + f([m - D^*(x)]^+)}{2}$.

We observe that $m \leq D^*(x) + [m - D^*(x)]^+$ and f is increasing, so with Equation (6.17), $2f(m/2) \leq V(G^*, D^*)$.

We show that $p_{G^*} = p_{\text{data}}$ (almost everywhere).

We showed that $2f(m/2) \leq V(G^*, D^*) \leq 2f(m/2)$, so $V(G^*, D^*) = 2f(m/2)$.

The equality occurs only if Equation (6.17) is an equality, and this happens only if $f(m/2) = \frac{f(D^*(x)) + f([m - D^*(x)]^+)}{2}$ for almost all x (and actually for all x since f and D^* are continuous). Let $a = \min_x D^*(x)$. Because of Lemma 6.1, we know that $a \in [0, m]$. Without loss of generality (the other case is symmetrical), we assume that $a \leq m - a$. Since f is convex, the necessary equality constraint is true only if f is affine on the interval $[a, m - a]$ (this is necessary but not sufficient). So there exists $\alpha > 0$ and $\beta \in \mathbb{R}$ such that $\forall x \in [a, m - a], f(x) = \alpha x + \beta$. Let h_η be the hat function of width 2η, i.e. $h_\eta(x) = \left[\frac{1}{\eta} - \frac{|x|}{\eta^2}\right]^+ 0$. We define

$$r(x) = \begin{cases} \frac{p_{G^*}(x)}{p_{G^*}(x) + p_{\text{data}}(x)} & \text{if } p_{\text{data}}(x) + p_{G^*}(x) \neq 0 \\ 1 & \text{otherwise} \end{cases} \tag{6.18}$$

Let $\eta > 0$, and $r_\eta(x) = (r * h)(x)$ where $*$ is the convolution operator. We note that r_η is continuous. We now define

$$\widetilde{D}_\eta(x) = (m - 2a) r_\eta(x) + a \tag{6.19}$$

We notice that $\forall x, \widetilde{D}_\eta(x) \in [a, m - a]$, so f is affine on the image of $\widetilde{D}_\eta$. Putting

$\widetilde{D}_\eta$ into equation (6.5), we obtain:

$$
\begin{aligned}
V(G^*, D^*) \leq & V(G^*, \widetilde{D}_\eta) && (6.20)\\
2f(m/2) \leq & \int_x p_{\text{data}}(x) f\left(\widetilde{D}_\eta(x)\right) \mathrm{d}x + \int_x p_{G^*(x)} f\left(\left[m - \widetilde{D}_\eta(x)\right]^+\right) \mathrm{d}x && (6.21)\\
\alpha m + 2\beta \leq & \alpha \int_x p_{\text{data}}(x) \widetilde{D}_\eta(x) \mathrm{d}x + \beta + \alpha \int_x p_{G^*(x)} \left(m - \widetilde{D}_\eta(x)\right) \mathrm{d}x + \beta && (6.22)\\
m \leq & \int_x (p_{\text{data}}(x) - p_{G^*}(x)) \widetilde{D}_\eta(x) \mathrm{d}x && (6.23)\\
0 \leq & \alpha(m - 2a) \int_x (p_{\text{data}}(x) - p_{G^*}(x)) r_\eta(x) \mathrm{d}x + 0 && (6.24)\\
0 \leq & \int_x (p_{\text{data}}(x) - p_{G^*}(x)) r_\eta(x) \mathrm{d}x. && (6.25)
\end{aligned}
$$

We notice that for all x, $r_\eta(x) < 1$, and for almost all x such that $p_{G^*}(x) < p_{\text{data}}(x)$ there exists η such that $r_\eta(x) < \frac{1}{2}$. We split the integral:

$$
\begin{aligned}
0 \leq & \frac{1}{2} \int_x \mathbb{1}_{p_{G^*}(x) < p_{\text{data}}(x)} (p_{\text{data}}(x) - p_{G^*}(x)) \mathrm{d}x \\
& + \int_x (1 - \mathbb{1}_{p_{G^*}(x) < p_{\text{data}}(x)}) (p_{\text{data}}(x) - p_{G^*}(x)) \mathrm{d}x && (6.26)\\
0 \leq & -\frac{1}{2} \int_x \mathbb{1}_{p_{G^*}(x) < p_{\text{data}}(x)} (p_{\text{data}}(x) - p_{G^*}(x)) \mathrm{d}x + 0 && (6.27)\\
0 \geq & \int_x \mathbb{1}_{p_{G^*}(x) < p_{\text{data}}(x)} (p_{\text{data}}(x) - p_{G^*}(x)) \mathrm{d}x && (6.28)
\end{aligned}
$$

so $\int_x \mathbb{1}_{p_{G^*}(x) < p_{\text{data}}(x)} (p_{\text{data}}(x) - p_{G^*}(x)) \mathrm{d}x = 0$ since the term in the integral is nonnegative. Thus, $p_{G^*}(x) \geq p_{\text{data}}(x)$ for almost all x. Since p_{G^*} and p_{data} are probability densities, this means that $p_{G^*} = p_{\text{data}}$ almost everywhere (this is proven in detail in Technical Lemma 6.1. □

Technical lemma 6.1 (used in proof of Theorem 6.1)**.** *If p and q are probability densities, then $\int_x \mathbb{1}_{p(x) < q(x)} \mathrm{d}x = 0$ if and only if $\int_x \mathbb{1}_{p(x) \neq q(x)} \mathrm{d}x = 0$.*

Proof. Let's assume that $\int_x \mathbb{1}_{p(x)<q(x)}\mathrm{d}x = 0$, i.e. for almost all x, $\mathbb{1}_{p(x)<q(x)} = 0$. Then

$$\begin{aligned}
& \int_x \mathbb{1}_{p(x)>q(x)}(p(x)-q(x))\mathrm{d}x && (6.29)\\
= & \int_x (1-\mathbb{1}_{p(x)\leq q(x)})(p(x)-q(x))\mathrm{d}x && (6.30)\\
= & \int_x p(x)\mathrm{d}x - \int_x q(x)\mathrm{d}x + \int_x \mathbb{1}_{p(x)\leq q(x)}(p(x)-q(x))\mathrm{d}x && (6.31)\\
= & 1-1+\int_x \left(\mathbb{1}_{p(x)<q(x)} + \mathbb{1}_{p(x)=q(x)}\right)(p(x)-q(x))\mathrm{d}x && (6.32)\\
= & \int_x \underbrace{\mathbb{1}_{p(x)<q(x)}}_{\substack{=0\\ \text{almost everywhere}}} (p(x)-q(x))\mathrm{d}x + \int_x \mathbb{1}_{p(x)=q(x)} \underbrace{(p(x)-q(x))}_{=0}\mathrm{d}x && (6.33)
\end{aligned}$$

So $\int_x \mathbb{1}_{p(x)>q(x)}(p(x)-q(x))\mathrm{d}x = 0$ and since the term in the integral is always non-negative, $\mathbb{1}_{p(x)>q(x)}(p(x)-q(x)) = 0$ for almost all x. And $p(x)-q(x)=0$ implies $\mathbb{1}_{p(x)>q(x)} = 0$, so $\mathbb{1}_{p(x)>q(x)} = 0$ almost everywhere. Therefore $\int_x \mathbb{1}_{p(x)>q(x)}\mathrm{d}x = 0$ which completes the proof, given the hypothesis. □

We gave necessary conditions on the Nash equilibriums. We can actually give necessary and sufficient conditions, so that we can fully characterize the Nash equilibriums. We also prove the existence of such a Nash equilibrium.

Theorem 6.2. *Let $v \in [0, m/2]$ be the largest number such that f is affine on $[m/2-v, m/2+v]$. Nash equilibriums are characterized by*

(a) $p_{G^*} = p_{\text{data}}$ (almost everywhere);

(b) there exists a constant $\gamma \in [m/2-v, m/2+v]$ that satisfies $D^*(x) = \gamma$ for all x such that $p_{\text{data}}(x) \neq 0$, and $D^*(x) \geq \gamma$ for all x such that $p_{\text{data}}(x) = 0$.

Proof.

We show the sufficient conditions.

Let (G^*, D^*) a pair that satisfies the two conditions of Theorem 6.2.

First we show that Equation (6.5) is satisfied. Let D be any discriminator.

$$\begin{aligned} V(G^*, D) \geq & V(G^*, \widehat{D}) && (6.34)\\ = & \int_x p_{\text{data}}(x) f\left(\widehat{D}(x)\right) \mathrm{d}x + \int_x p_{G^*}(x) f\left(\left[m - \widehat{D}(x)\right]^+\right) \mathrm{d}x && (6.35)\\ = & \int_x p_{\text{data}}(x) \left(f\left(\widehat{D}(x)\right) + f\left(\left[m - \widehat{D}(x)\right]^+\right)\right) \mathrm{d}x && (6.36)\\ \geq & 2f(m/2) && (6.37) \end{aligned}$$

where Equation (6.34) comes from Lemma 6.1 and Equation (6.37) comes from the convexity of f. We now compute

$$\begin{aligned} V(G^*, D^*) = & \int_x p_{\text{data}}(x) \left(f\left(D^*(x)\right) + f\left(\left[m - D^*(x)\right]^+\right)\right) \mathrm{d}x && (6.38)\\ = & \int_x p_{\text{data}}(x) \left(f\left(\gamma\right) + f\left(m - \gamma\right)\right) \mathrm{d}x && (6.39) \end{aligned}$$

which is true since either $p_{\text{data}}(x) = 0$ or $D^*(x) = \gamma$ (almost everywhere). Since f is affine on $[\gamma, m - \gamma]$, we obtain that $V(G^*, D^*) = 2f(m/2)$. So $V(G^*, D) \geq V(G^*, D^*)$, which is Equation (6.5).

It is easy to show that Equation (6.6) is satisfied, for any generator G:

$$U(G, D^*) = \int_x p_G(x) f\left(D^*(x)\right) \mathrm{d}x \tag{6.40}$$

$$= \int_x \mathbb{1}_{p_{\text{data}}(x)=0} p_G(x) f\left(D^*(x)\right) \mathrm{d}x + \int_x \mathbb{1}_{p_{\text{data}}(x)\neq 0} p_G(x) f(\gamma) \mathrm{d}x \tag{6.41}$$

$$\geq \int_x \mathbb{1}_{p_{\text{data}}(x)=0} p_G(x) f(\gamma) \mathrm{d}x + \int_x \mathbb{1}_{p_{\text{data}}(x)\neq 0} p_G(x) f(\gamma) \mathrm{d}x \tag{6.42}$$

$$= f(\gamma) \tag{6.43}$$

and $U(G^*, D^*) = \int_x p_{\text{data}}(x) f\left(D^*(x)\right) \mathrm{d}x = f(\gamma)$. So $U(G, D^*) \geq U(G^*, D^*)$.

We now show the necessary conditions.

Let (G^*, D^*) be a Nash equilibrium. We know from Theorem 6.1 that the first condition of Theorem 6.2 is true. We now show condition (b). We negate the condition (b) and find a contradiction. Let us first assume that D^* is not constant where p_{data} is non-zero. Since D^* is nontinuous, there exists x_0 and x_1 such that $D^*(x_0) \neq D^*(x_1)$ and p_{data} is non-zero on a neighborhood of x_0 and x_1. Without loss of generality, we can assume that $D^*(x_0) < D^*(x_1)$. Let $C = \frac{D^*(x_0)+D^*(x_1)}{2}$. We define $\mathcal{S}^-$ and $\mathcal{S}^+$ by:

$$\mathcal{S}^- = \{x | D^*(x) < C \text{ and } p_{\text{data}}(x) \neq 0\} \tag{6.44}$$

$$\mathcal{S}^+ = \{x | D^*(x) \geq C \text{ and } p_{\text{data}}(x) \neq 0\}. \tag{6.45}$$

By assumption, both sets are non empty and have non-zero measure. We define the generator G_0, defined by

$$G_0(x) = \begin{cases} K p_{\text{data}}(x) & \text{if } x \in \mathcal{S}^- \\ 0 & \text{otherwise} \end{cases} \tag{6.46}$$

Where $K = \frac{1}{\int_{\mathcal{S}^-} p_{\text{data}}(x)\mathrm{d}x}$. The denominator is non-zero and smaller than 1, so K is well defined and greater than 1. We compute

$$U(G^*, D^*) - U(G_0, D^*) \tag{6.47}$$

$$= \int_x (p_{data} - p_{G_0}) f\left(D^*(x)\right) \mathrm{d}x \tag{6.48}$$

$$= \int_x (p_{data} - p_{G_0})(D^*(x) - C)\mathrm{d}x \tag{6.49}$$

$$= \int_{\mathcal{S}^-} (p_{\text{data}}(x) - p_{G_0}(x))(D^*(x) - C)\mathrm{d}x + \int_{\mathcal{S}^+} (p_{\text{data}}(x) - p_{G_0}(x))(D^*(x) - C)\mathrm{d}x \tag{6.50}$$

$$= \int_{\mathcal{S}^-} \underbrace{(p_{\text{data}}(x) - K p_{\text{data}}(x))}_{<0} \underbrace{(D^*(x) - C)}_{<0} \mathrm{d}x + \int_{\mathcal{S}^+} \underbrace{p_{\text{data}}(x)}_{\geq 0} \underbrace{(D^*(x) - C)}_{\geq 0} \mathrm{d}x \tag{6.51}$$

The left term is strictly positive (since $\mathcal{S}^-$ has non-zero measure) and the right term is nonnegative. So $U(G^*, D^*) > U(G_0, D^*)$, which which violates Equation (6.6).

We now assume that D^* is constant where p_{data} is non-zero. Let γ be this constant. We first show that this constant is in $[m/2 - v, m/2 + v]$. Let us assume it is not in this interval. We know that $\gamma \in [0, m]$. We know that $p_{G^*} = p_{\text{data}}$, so we can compute:

$$V(G^*, D^*) = \int_x \big(p_{\text{data}}(x) f(D^*(x)) + p_{G^*}(x) f([m - D^*(x)]^+)\big)\mathrm{d}x \tag{6.52}$$

$$= \int_x \big(p_{\text{data}}(x) f(D^*(x)) + p_{\text{data}}(x) f(m - D^*(x))\big)\mathrm{d}x \tag{6.53}$$

$$= \int_x \mathbb{1}_{p_{\text{data}}(x) \neq 0}\ p_{\text{data}}(x)\big(f(\gamma) + f(m - \gamma)\big)\mathrm{d}x + 0 \tag{6.54}$$

$$= f(\gamma) + f(m - \gamma) \tag{6.55}$$

and by definition of v, f is strictly convex on at least a part of $[\gamma, m-\gamma]$, so we $f(\gamma) + f(m-\gamma) > 2f(m/2)$. However $V(G^*, D_{m/2}) = 2f(m/2)$, (where $D_{m/2}$ is the discriminator that produces the constant value $m/2$). This is in contradiction with the optimality of D^*.

Finally, we show that $D^*(x) \geq \gamma$ for all x such that $p_{\text{data}}(x) = 0$. Let us assume that there exists x_0 such that $p_{\text{data}}(x_0) = 0$ and $D^*(x_0) < \gamma$. Since D^* is continuous, there exists a neighborhood A of x_0 such that $D^*(x) < \gamma$ for all $x \in A$. Moreover, since $D^*(x) = \gamma$ for all x such that $p_{\text{data}}(x) \neq 0$, we must have $p_{\text{data}}(x) = 0$ for all $x \in A$. We now define G_A the generator that has density $\frac{\mathbb{1}_{x \in A}}{|A|}$ (where $|A| = \int_x \mathbb{1}_{x \in A} \mathrm{d}x$). We already know that G^* is the generator that generates p_{data}, so we can compute

$$
\begin{aligned}
U(G_A, D^*) &= \int_x \frac{\mathbb{1}_{x \in A}}{|A|} f(D^*(x)) \mathrm{d}x && (6.56)\\
&< \int_x \frac{\mathbb{1}_{x \in A}}{|A|} f(\gamma) \mathrm{d}x && (6.57)\\
&= f(\gamma) && (6.58)\\
&= U(G^*, D^*) && (6.59)
\end{aligned}
$$

This is in contradiction with Equation (6.6). □

The previous theorems are difficult to interpret at first glance. However, under mild assumptions (f strictly convex and p_{data} non-zero almost everywhere), they simplify into something intuitive.

Corollary 6.1. *If f is strictly convex and $p_{\text{data}} \neq 0$ almost everywhere, then (D^*, G^*) is a Nash equilibrium if and only if $p_{G^*} = p_{\text{data}}$ and $D^*(x) = m/2$ for all $x \in \Omega$.*

The assumption of Corollary 6.1 that $p_{\text{data}} \neq 0$ almost everywhere is generally not true, in particular under the manifold assumption (section 1.2). However, we can add Gaussian noise, even with very small variance, to the input of the discriminator in order to achieve this property [Arjovsky and Bottou, 2017]. We can also formulate another corollary if we do not make this assumption.

Corollary 6.2. *If f is strictly convex, then (D^*, G^*) is a Nash equilibrium if and only if $p_{G^*} = p_{\text{data}}$ and $D^*(x) = m/2$ for all $x \in \Omega$ such that $p_{\text{data}}(x) \neq 0$ and $D^*(x) \geq m/2$ elsewhere.*

6.2.3 Auto-encoders as discriminator

In the experiments, the discriminator D is structured as an auto-encoder:

$$D(x) = ||\text{Dec}(\text{Enc}(x)) - x||_2^2. \tag{6.60}$$

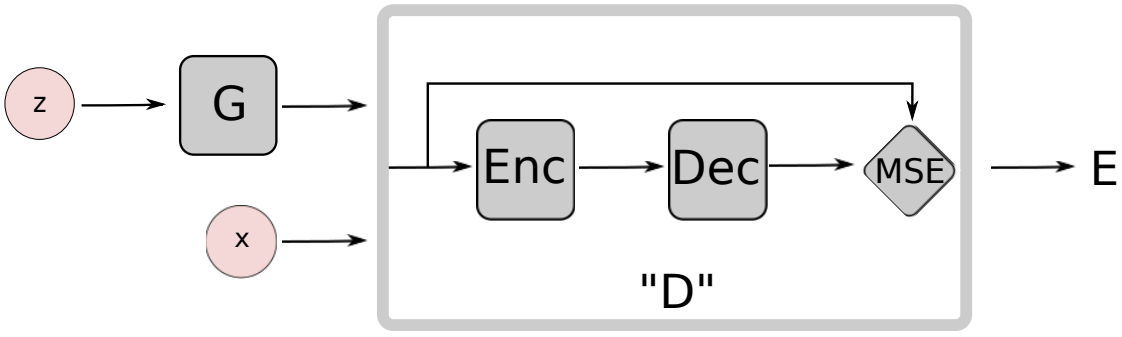


Figure 6.1: EBGAN architecture with an auto-encoder discriminator.

The diagram of the EBGAN model with an auto-encoder discriminator is depicted in Figure 6.1. The choice of the auto-encoders for D may seem arbitrary at the first glance, yet we postulate that it is conceptually more attractive than a binary logistic network:

- Rather than using a single bit of target information to train the model, the reconstruction-based output offers a diverse targets for the discriminator. With the binary logistic loss, only two targets are possible, so within a mini-batch, the gradients corresponding to different samples are most likely far from orthogonal. This leads to inefficient training, and reducing the mini-batch sizes is often not an option on current hardware. On the other hand, the reconstruction loss will likely produce very different gradient directions within the minibatch, allowing for larger minibatch size without loss of efficiency.

- Auto-encoders have traditionally been used to represent energy-based model and arise naturally. When trained with some regularization terms (see section 6.2.3.1), auto-encoders have the ability to learn an energy manifold without supervision or negative examples. This mean that even when an EBGAN auto-encoding model is trained to reconstruct a *real* sample, the discriminator contributes to discovering the data manifold by itself. To the contrary, without the presence of negative examples from the generator, a discriminator trained with binary logistic loss becomes pointless.

6.2.3.1 Connection to the regularized auto-encoders

One common issue in training auto-encoders is that the model may learn little more than an identity function. From an energy-based perspective, this means attributing zero energy to the whole space. In order to avoid this problem, the model must be pushed to give higher energy to points outside the data manifold. Theoretical and experimental results have addressed this issue by regularizing the latent representations [Vincent et al., 2010, Rifai et al., 2011, Poultney et al., 2006,

Kavukcuoglu et al., 2010b]. Such regularizers aim at restricting the reconstructing power of the auto-encoder so that it can only attribute low energy to a smaller portion of the input points.

We argue that the energy function (the discriminator) in the EBGAN framework can also be seen as being regularized by having a generator producing the contrastive samples, to which the discriminator ought to give high reconstruction energies. We further argue that the EBGAN framework allows more flexibility from this perspective, because: (i)-the regularizer (generator) is fully trainable instead of being handcrafted; (ii)-the adversarial training paradigm enables a direct interaction between the processes of producing contrastive sample and learning the energy function.

Furthermore, recent work such as [Larsen et al., 2015] addresses the insufficient capacity of the ℓ_2 loss function; the authors show that training a variational auto-encoder with the element-wise ℓ_2 loss failed to capture the fine details in its reconstruction. However, we argue that the EBGAN framework is established from an orthogonal angle where the ℓ_2 loss function (or any other loss in the energy-based scope) merely serves to produce an energy. It is not a problem if the discriminator of an EBGAN does not reconstruct perfectly. The bottom line is that the discriminator is able to distinguish real and fake images by predicting energies.

6.2.4 Repelling regularizer

We propose a "repelling regularizer" which fits well into the EBGAN auto-encoder model, to keep the model from producing samples that are clustered in one or only few modes of p_{data}. Another technique "minibatch discrimination" was

developed by [Salimans et al., 2016] from the same philosophy.

We implement repelling regularizer, or Pulling-away Term (PT), at a representation level. Formally, let $S \in \mathbb{R}^{s \times N}$ denotes a batch of sample representations taken from the encoder output layer. The PT term is defines as:

$$f_{PT}(S) = \frac{1}{N(N-1)} \sum_{i} \sum_{j \neq i} \Big(\frac{S_i^{\mathsf{T}} S_j}{\|S_i\| \|S_j\|} \Big)^2. \tag{6.61}$$

The PT term attempts to decrease the magnitude of cosine similarity between pairwise sample representations, and thus making them as orthogonal as possible. Prior work showed that the output layer of encoder carries representational powerful information for various tasks [Rasmus et al., 2015, Zhao et al., 2016a]. The rationale for choosing the cosine similarity instead of Euclidean distance is to make the term bounded below and invariant to scale. We use the notation "EBGAN-PT" to refer to the EBGAN auto-encoder model trained with this PT term. Note the PT term is used in the generator loss but not in the discriminator loss.

6.3 Experiments

This section presents experimental results and empirical validation of the EBGAN framework. For implementation details, we refer to [Zhao et al., 2016b].

6.3.1 Exhaustive grid search on MNIST

In this section we demonstrate the better training stability of EBGANs over GANs on a simple task of MNIST digit generation with fully-connected networks. We run an exhaustive grid search over a set of architectural choices and hyper-

parameters for both frameworks. The convolutional architectures applied on larger scale and more complex datasets are exhibited in later sections. Formally, we specify the search grid in Table 6.1. `nLayerD` represents the total number of layers of Enc and Dec put together.

To analyze the results, we use a variant of the *inception score* [Salimans et al., 2016] as a numerical means for assessing generation quality. In order to evaluate the quality of MNIST generations, we pretrain a classifier on the full labeled MNIST dataset. Let $p_c(C = c|X = x)$ the estimated probability (by the trained classifier) that the sample x has the label c. We defined the tweaked inception score of a generator G is given by

$$I' = \mathbb{E}_{x\sim p_G}\left[D_{KL}\left(\int_{x'\in\Omega} p_c(C = c|X = x')p_G(X = x')\mathrm{d}x'\middle\|p_c(C = c|X = x)\right)\right]. \tag{6.62}$$

It differs from the original inception score by two aspects: the two terms in the KL-divergence have been swapped and the exponential has been dropped. The reason to both these changes are to produce more compact, easily readable histograms. Consequently, higher I' score implies better quality of the generation.

It is also worth noting that although we inherit the name "inception score" from [Salimans et al., 2016], the evaluation isn't related to the "inception" model trained on ImageNet dataset. The classifier is a regular 3 layer convolutional network trained on MNIST.

We plot the histogram of I' scores in Figure 6.2. We further separated out the optimization related setting from GAN's grid (`optimD`, `optimG` and `lr`) and plot the histogram of each sub-grid individually, together with the EBGAN scores as a reference, in Figure 6.3. The number of experiments for GANs and EBGANs are

Settings	Description	EBGANs	GANs
`nLayerG`	number of layers in G	[2, 3, 4, 5]	[2, 3, 4, 5]
`nLayerD`	number of layers in D	[2, 3, 4, 5]	[2, 3, 4, 5]
`sizeG`	size of layers in G	[400, 800, 1600, 3200]	[400, 800, 1600, 3200]
`sizeD`	size of layers in D	[128, 256, 512, 1024]	[128, 256, 512, 1024]
`dropoutD`	dropout in D?	[true, false]	[true, false]
`optimD`	optimizer for D	adam	[adam, sgd]
`optimG`	optimizer for G	adam	[adam, sgd]
`lr`	learning rate	0.001	[0.01, 0.001, 0.0001]

Table 6.1: MNIST grid search parameters

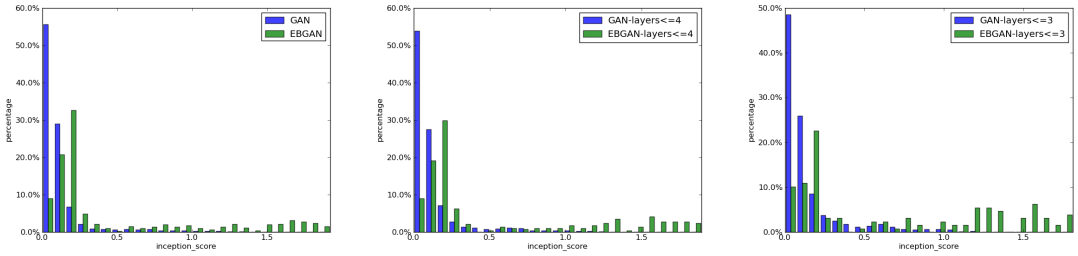


Figure 6.2: Histogram of the tweaked inception scores from the grid search. The x-axis corresponds to the inception score I'. Higher score means better generations. Left (a): general comparison of EBGANs (green) versus GANs (blue); Middle (b): EBGANs and GANs both constrained by `nLayer[GD]<=4`; Right (c): EBGANs and GANs both constrained by `nLayer[GD]<=3`.

both 512 in every subplot. The histograms are intended to show that EBGANs are generally more reliably trained than GANs. We can see that GANs require much more careful tuning: only a very small set of parameters produces large inception scores. On the other hand, EBGANs have a higher success rate, which means more flexibility in the converging parameters.

Finally, digits generated from the configurations presenting the best inception score are shown in Figure 6.4.

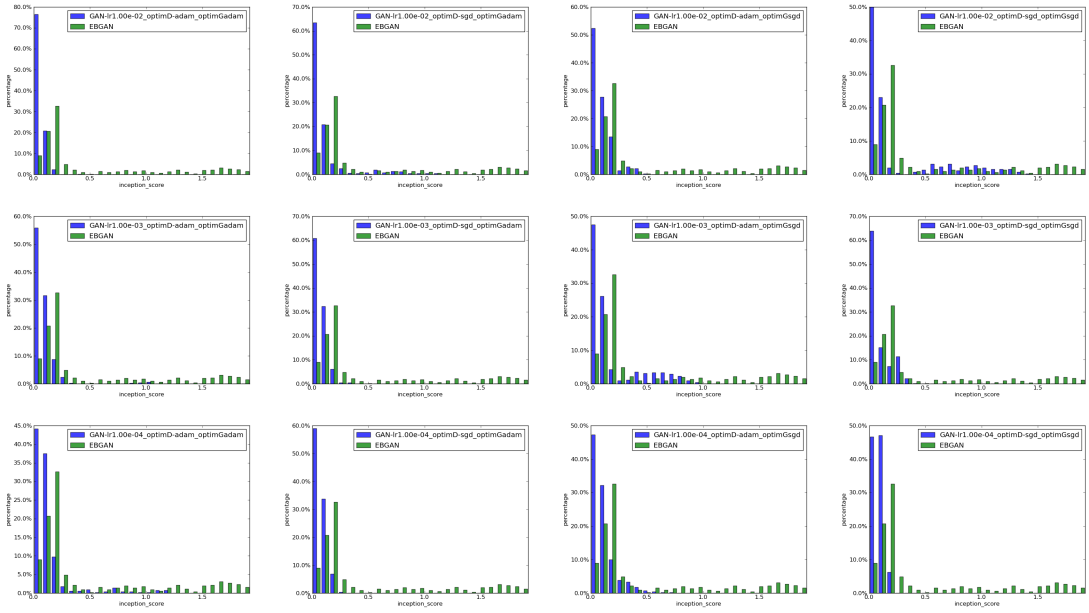


Figure 6.3: Histogram of the tweaked inception scores grouped by different optimization combinations, drawn from `optimD`, `optimG` and `lr`.

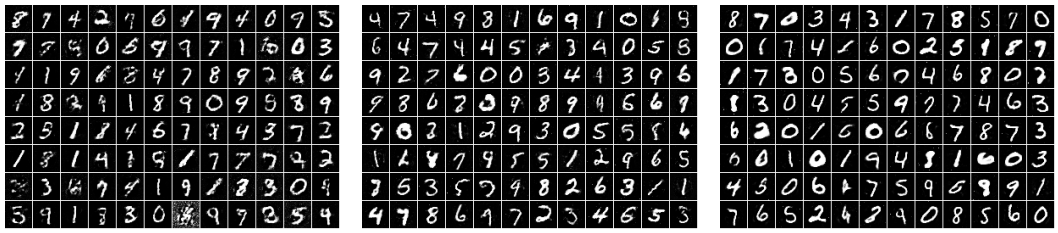


Figure 6.4: Generation from the grid search on MNIST.
In all cases, `nLayerG` $= 5$, `nLayerD` $= 2$, `sizeD` $= 2014$, `dropoutD` $=$ false.
Left(a): Best GAN: `sizeG` $= 1600$, `optimD` $=$ `optimG` $=$ SGD, `lr` $= 0.01$.
Middle(b): Best EBGAN: `sizeG` $= 800$, `optimD` $=$ `optimG` $=$ ADAM,
`lr` $= 0.001$, `margin` $= 10$.
Right(c): Best EBGAN-PT: `sizeG` $= 800$, `optimD` $=$ `optimG` $=$ ADAM,
`lr` $= 0.001$, `margin` $= 10$, $\lambda_{PT} = 0.1$.

model	100	200	1000
Ladder Network [Pezeshki et al., 2015]	1.69±0.18	-	1.05±0.02
Ladder Network [Rasmus et al., 2015]	1.09±0.32	-	0.90±0.05
Ladder Network (custom implementation)	1.36±0.21	1.24±0.09	1.04±0.06
Ladder Network (custom) + EBGAN	**1.04±0.12**	**0.99±0.12**	**0.89±0.04**

Table 6.2: The comparison of Ladder Network model and its EBGAN extension on PI-MNIST semi-supervised task. The results are error rate (in %) and are averaged over 15 different random seeds.

6.3.2 Semi-supervised learning on MNIST

We explore the potential of using the EBGAN framework for semi-supervised learning on permutation-invariant MNIST, collectively on using 100, 200 and 1000 labels. We utilized a bottom-layer-cost Ladder Network (LN) [Rasmus et al., 2015] with the EGBAN framework, which we call EBGAN-LN. Ladder Network can be categorized as an energy-based model that is built with both feedforward and feedback hierarchies with powerful lateral connections coupling two pathways.

Table 6.2 shows the semi-supervised results on the PI-MNIST. We observe that positioning a bottom-layer-cost Ladder Network into an EBGAN framework profitably improves the performance of the Ladder Network itself. We postulate that the contrastive samples produced by the generator acts as an effective regularizer. Ladder Networks are still difficult to train, as the discrepancy between the reported results between [Rasmus et al., 2015] and [Pezeshki et al., 2015] show. In order to add the EBGAN framework to Ladder Networks, we had to reimplement it and the results we obtained are between those of [Rasmus et al., 2015] and those of [Pezeshki et al., 2015].

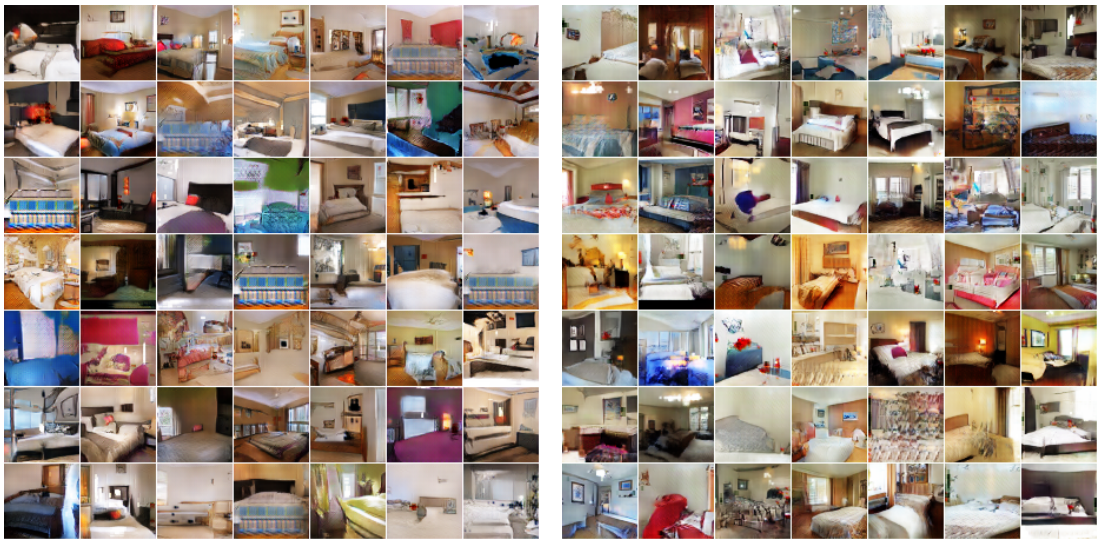


Figure 6.5: Generation from LSUN bedroom dataset. Left(a): DCGAN generation. Right(b): EBGAN-PT generation.

6.3.3 LSUN & CelebA

We apply the EBGAN framework with deep convolutional architecture to generate 64×64 RGB images, a more realistic task, using the LSUN bedroom dataset [Yu et al., 2015] and the large-scale face dataset CelebA under alignment [Liu et al., 2015]. To compare EBGANs with DCGANs [Radford et al., 2015], we train a DCGAN model under the same configuration and show its generation side-by-side with the EBGAN model, in Figures 6.5 and 6.6.

While both frameworks have produced high quality generations using LSUN bedroom pictures, EBGAN-PTs seem to cover richer modes and successfully exclude some recurring off-manifold parts, such as the blue bed-ish shape appearing frequently at the bottom of the generations in Figure 6.5(a). The repelling regularizer term (PT) of EBGAN-PT therefore seems to effectively help with the mode collapse issue.

We also provide generations on an augmented version of the LSUN bedroom

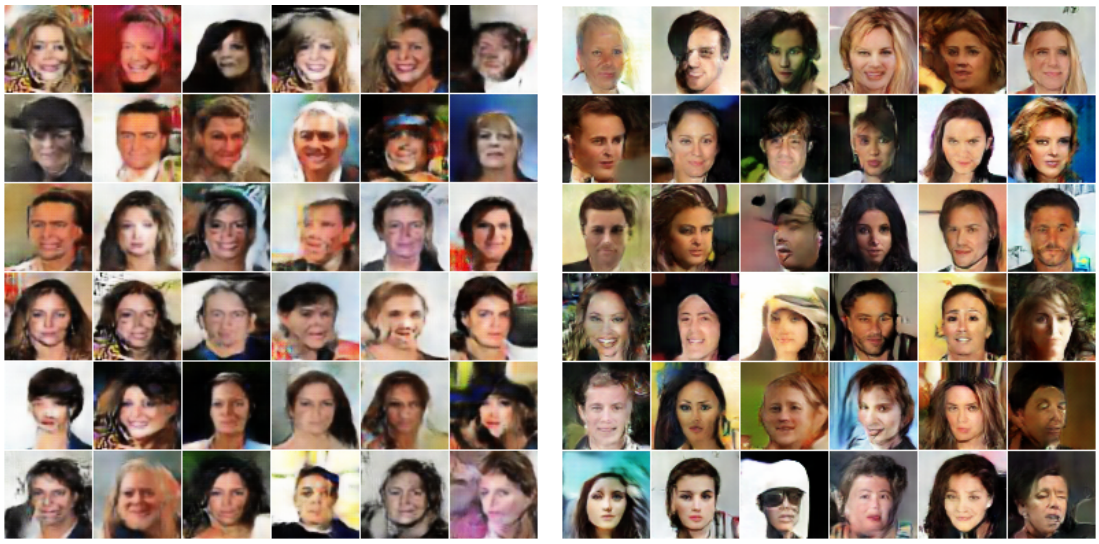


Figure 6.6: Generation from CelebA face dataset. Left(a): DCGAN generation. Right(b): EBGAN-PT generation.

dataset. Instead of rescaling the original 96×96 images into 64×64 patches, this dataset is composed of randomly selected 64×64 patches (without rescaling). The generations are showed on Figure 6.7.

6.3.4 ImageNet

Finally, we trained EBGANs to generate high-resolution images on ImageNet [Russakovsky et al., 2015]. Compared with the datasets we have experimented so far, ImageNet presents an extensively larger and wilder space, so modeling the data distribution by a generative model becomes very challenging. We generate 128×128 images, trained on the full ImageNet-1k dataset, which contains roughly 1.3 million images from 1000 different categories. We also trained a network to generate images of size 256×256, on a dog-breed subset of ImageNet. The results are shown in Figures 6.8 and 6.9. Despite the difficulty of generating images on a high-resolution level, we observe that EBGANs are able to learn about the fact that

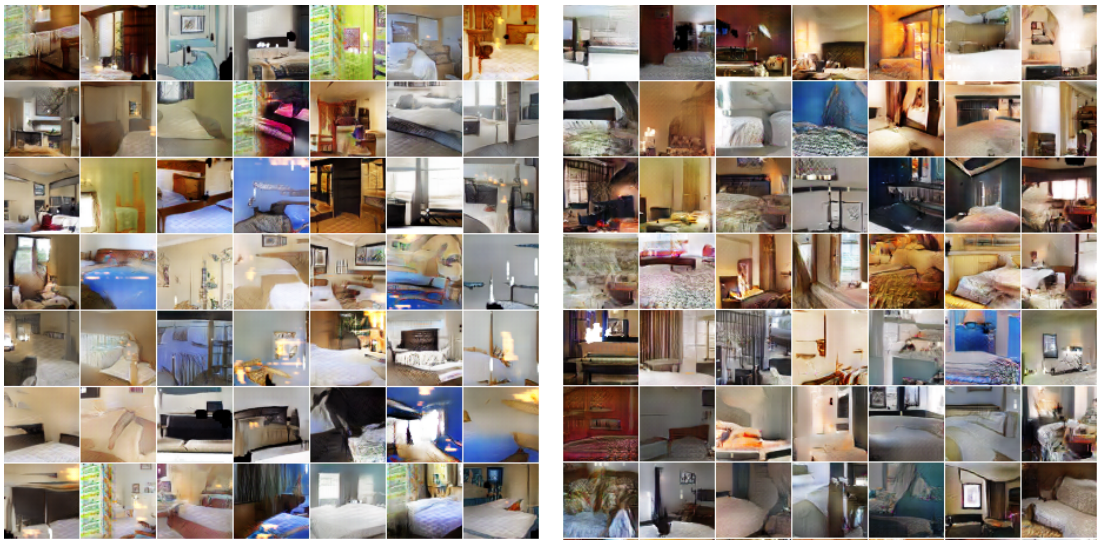


Figure 6.7: Generation from the augmented LSUN bedroom dataset. Left(a): DCGAN generation. Right(b): EBGAN-PT generation.

objects appear in the foreground, together with various background components resembling grass texture, sea under the horizon, mirrored mountain in the water, buildings, etc. In addition, our 256×256 dog-breed generations, although far from realistic, do reflect some knowledge about the appearances of dogs such as their body, furs and eye.

6.3.5 Comparison of EBGANs and EBGAN-PTs

To further demonstrate how the pull-away term may influence EBGAN auto-encoder training, we chose LSUN bedrooms, both whole-image and augmented-patch version, and CelebA to make further demonstration. The comparison of EBGAN and EBGAN-PT generation are showed in Figure 6.10, Figure 6.11 and Figure 6.12. Note that all comparison pairs adopt identical architectural and hyper-parameter setting as used in section 6.3.3. The generations with the pulling-away term (PT) do not show obvious repeated patterns.

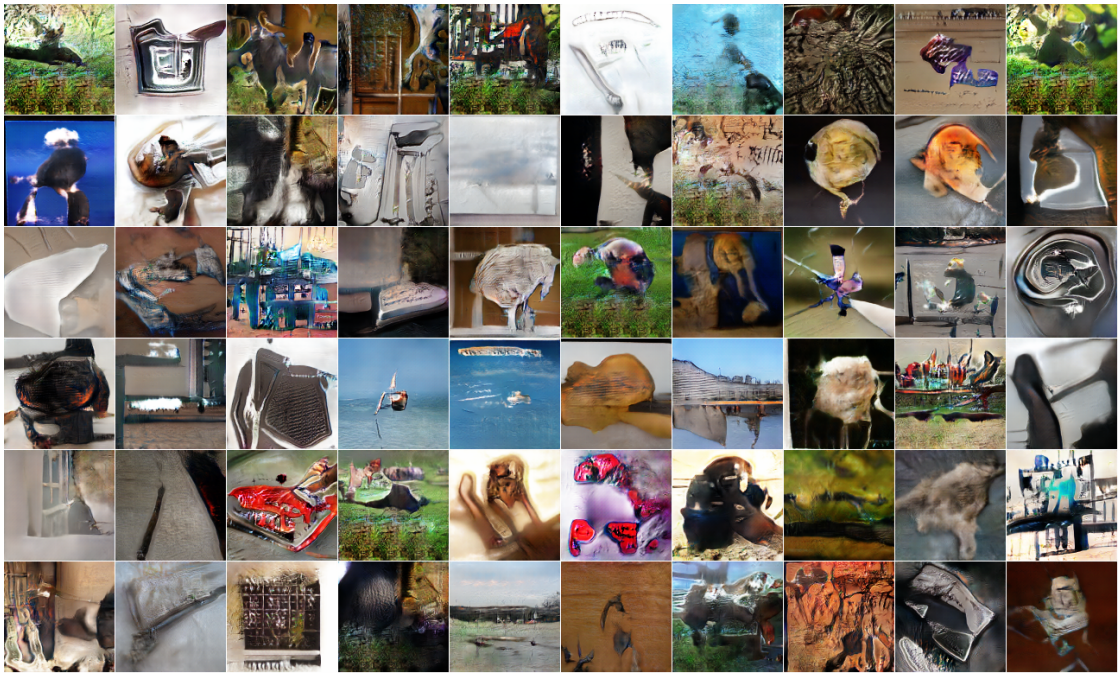


Figure 6.8: Sample generations from the 128 × 128 pixels ImageNet dataset using an EBGAN-PT.

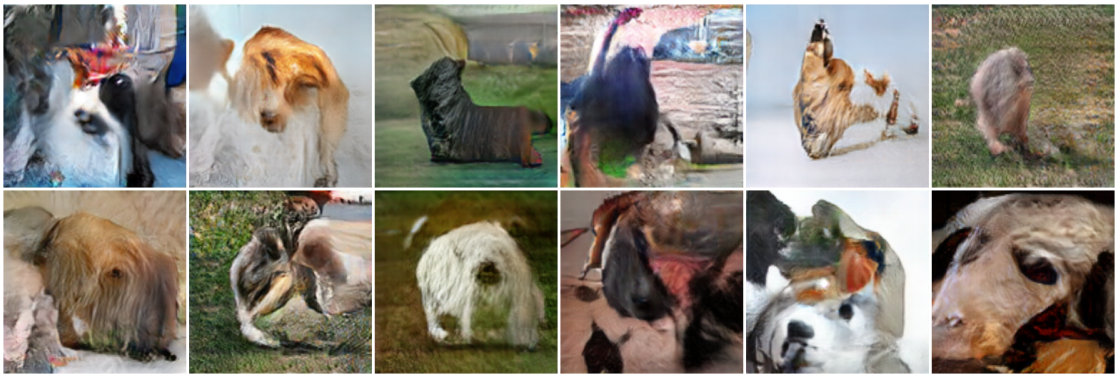
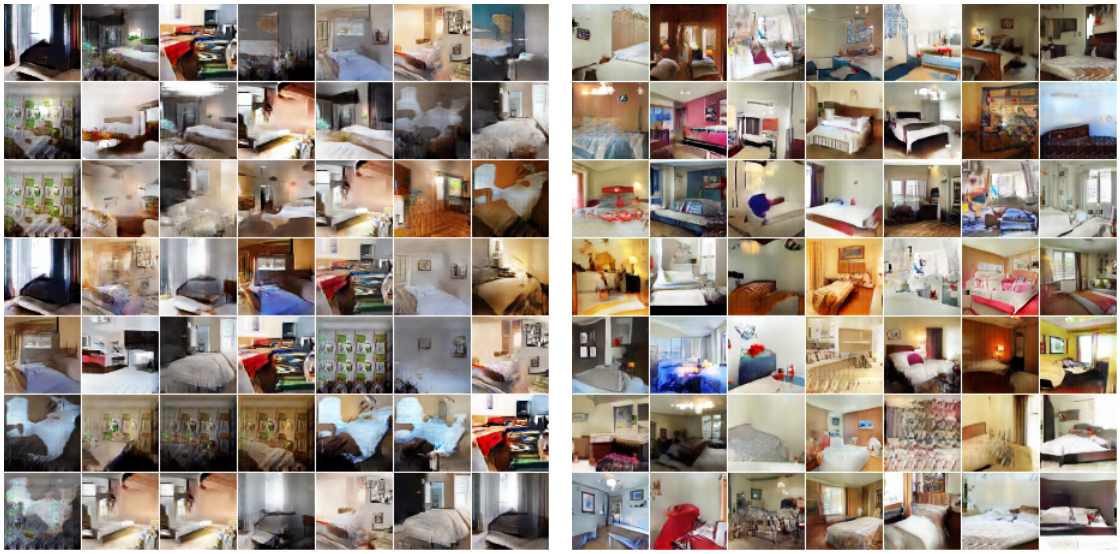


Figure 6.9: Sample generations from the 256 × 256 pixels ImageNet dataset restricted to dog breeds, using an EBGAN-PT.

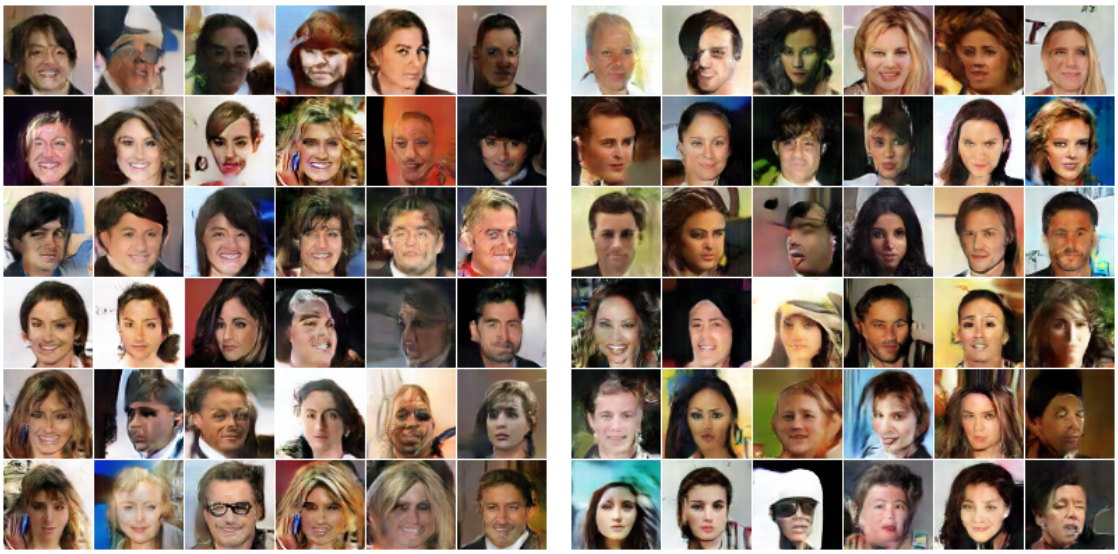


(a) EBGAN (b) EBGAN-PT

Figure 6.10: Comparison between EBGAN and EBGAN-PT on LSUN (resized to 64×64).

(a) EBGAN (b) EBGAN-PT

Figure 6.11: Comparison between EBGAN and EBGAN-PT on augmented LSUN (random 64×64 crops).



(a) EBGAN (b) EBGAN-PT

Figure 6.12: Comparison between EBGAN and EBGAN-PT on the CelebA dataset.

Chapter 7

Conclusion

We explored different methods to address uncertainty in video prediction. Chapter 3, published as [Goroshin et al., 2015b], uses a linearizing penalty, and latent variables, with a prediction task. Chapter 4 was published as [Mathieu et al., 2015], and introduced Conditional Generative Adversarial Networks as a trainable loss, with the ability to represent multimodal distributions. In Chapter 5, we combined Generative Adversarial Networks with Variational Auto-Encoders to disentangle factors of variation without data alignment [Mathieu et al., 2016]. Finally, Chapter 6, published as [Zhao et al., 2016b], introduced a new energy-based loss for Generative Adversarial Networks, and opened the way to another approach of GANs.

However, unsupervised learning remains rarely used in large-scale systems. It is still an ill-defined problem and supervised learning is often seen as a much lower hanging fruit. However, the ever-increasing computing power poses the question of the limitation of the size of labeled datasets. The progress of machine learning, in particular in the domain of natural language processing and robotics, opens

to way to algorithms with a sort of "common sense", which is not easily labeled. Therefore, unsupervised learning is viewed as a central topic of artificial intelligence research on the long term. It may interact with other types of learning, such as supervised learning, in the context of semi-supervised learning, and reinforcement learning. Predictive learning is seen as a relevant problem, both theoretically for the complexity of the task, and practically. Model-based reinforcement learning uses a model of the world as a way to predict what would happen if a certain action is taken, which helps the algorithm to reduce its variance and converge faster. However, pure predictive learning alone may be over-simplifying the problem, and the whole process would probably benefit from other forms of supervision in a joint learning setup.

Another critical issue is evaluation. Generative models are difficult to evaluate and rank. Unsupervised learning trains a model for a task that is not the end goal, but merely a proxy to it, and a ranking of the models on the proxy may not reflect correctly to the end goal. Besides, generative models such as GANs do not have an intrinsic performance measure, the loss of the discriminator or the generator do not reflect the quality of the generations. Future work is clearly needed in this domain, in particular identifying subtle versions of the Mode Collapse problem. Although novel research has improved the quality of unsupervised learning, it is likely that a future breakthrough will at least partly come from larger data. Looking back at the story of supervised learning, the major breakthrough did not come from a single better algorithm, although incremental progress had been made, but from the sheer size of the data and model. Unsupervised learning is considerably harder than supervised learning, mainly due to the complexity of the distributions to be modeled. Therefore, it is likely that even though unsupervised learning is

not so successful yet, research in this area should not be disregarded in favor of purely supervised learning, as it is probable that the computing power is not quite sufficient yet to unlock its full potential.

Bibliography

[Arjovsky and Bottou, 2017] Arjovsky, M. and Bottou, L. (2017). Towards principled methods for training generative adversarial networks.

[Arnold and Ollivier, 2012] Arnold, L. and Ollivier, Y. (2012). Layer-wise learning of deep generative models. *CoRR*, abs/1212.1524.

[Ascenso et al., 2005] Ascenso, J., Brites, C., and Pereira, F. (2005). Improving frame interpolation with spatial motion smoothing for pixel domain distributed video coding. In *5th EURASIP Conference on Speech and Image Processing, Multimedia Communications and Services*, pages 1–6.

[Bengio, 2009] Bengio, Y. (2009). Learning deep architectures for AI. *Foundations and trends in Machine Learning*, 2(1):1–127.

[Bengio et al., 2013] Bengio, Y., Courville, A., and Vincent, P. (2013). Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828.

[Bishop, 1994] Bishop, C. M. (1994). Mixture density networks.

[Bromley et al., 1993] Bromley, J., Bentz, J. W., Bottou, L., Guyon, I., LeCun, Y., Moore, C., Säckinger, E., and Shah, R. (1993). Signature verification using

a siamese time delay neural network. *Int. Journal of Pattern Recognition and Artificial Intelligence*, 7(04):669–688.

[Cadieu and Olshausen, 2012] Cadieu, C. F. and Olshausen, B. A. (2012). Learning intermediate-level representations of form and motion from natural movies. *Neural Computation*.

[Carreira-Perpinan and Hinton, 2005] Carreira-Perpinan, M. A. and Hinton, G. E. (2005). On contrastive divergence learning. In *AISTATS*, volume 10, pages 33–40.

[Cayton, 2005] Cayton, L. (2005). Algorithms for manifold learning.

[Cheung et al., 2014] Cheung, B., Livezey, J. A., Bansal, A. K., and Olshausen, B. A. (2014). Discovering hidden factors of variation in deep networks. *CoRR*, abs/1412.6583.

[Cohen and Welling, 2014] Cohen, T. S. and Welling, M. (2014). Transformation properties of learned visual representations. *arXiv preprint arXiv:1412.7659*.

[Collobert et al., 2011] Collobert, R., Kavukcuoglu, K., and Farabet, C. (2011). Torch7: A Matlab-like environment for machine learning. In *BigLearn, NIPS Workshop*.

[Dai et al., 2015] Dai, J., He, K., and Sun, J. (2015). Instance-aware semantic segmentation via multi-task network cascades. *CoRR*, abs/1512.04412.

[Deng et al., 2009] Deng, J., Dong, W., Socher, R., Li, L. J., Li, K., and Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255.

[Denton et al., 2015] Denton, E. L., Chintala, S., Szlam, A., and Fergus, R. (2015). Deep generative image models using a laplacian pyramid of adversarial networks. *CoRR*, abs/1506.05751.

[Donahue et al., 2014] Donahue, J., Hendricks, L. A., Guadarrama, S., Rohrbach, M., Venugopalan, S., Saenko, K., and Darrell, T. (2014). Long-term recurrent convolutional networks for visual recognition and description. *CoRR*, abs/1411.4389.

[Donahue et al., 2013] Donahue, J., Jia, Y., Vinyals, O., Hoffman, J., Zhang, N., Tzeng, E., and Darrell, T. (2013). Decaf: A deep convolutional activation feature for generic visual recognition. *CoRR*, abs/1310.1531.

[Donahue et al., 2016] Donahue, J., Krähenbühl, P., and Darrell, T. (2016). Adversarial feature learning. *CoRR*, abs/1605.09782.

[Dosovitskiy et al., 2014] Dosovitskiy, A., Springenberg, J. T., and Brox, T. (2014). Learning to generate chairs with convolutional neural networks. *CoRR*, abs/1411.5928.

[Dumoulin et al., 2016] Dumoulin, V., Belghazi, I., Poole, B., Lamb, A., Arjovsky, M., Mastropietro, O., and Courville, A. (2016). Adversarially learned inference. *arXiv preprint arXiv:1606.00704*.

[Edwards and Storkey, 2015] Edwards, H. and Storkey, A. (2015). Censoring representations with an adversary. *arXiv preprint arXiv:1511.05897*.

[Farabet et al., 2013] Farabet, C., Couprie, C., Najman, L., and LeCun, Y. (2013). Learning hierarchical features for scene labeling. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1915–1929.

[Flynn et al., 2015] Flynn, J., Neulander, I., Philbin, J., and Snavely, N. (2015). Deepstereo: Learning to predict new views from the world's imagery. *CoRR*, abs/1506.06825.

[Georghiades et al., 2001] Georghiades, A. S., Belhumeur, P. N., and Kriegman, D. J. (2001). From few to many: Illumination cone models for face recognition under variable lighting and pose. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 23(6):643–660.

[Girshick et al., 2013] Girshick, R. B., Donahue, J., Darrell, T., and Malik, J. (2013). Rich feature hierarchies for accurate object detection and semantic segmentation. *CoRR*, abs/1311.2524.

[Goodfellow et al., 2014] Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A. C., and Bengio, Y. (2014). Generative adversarial networks. *NIPS*.

[Goroshin et al., 2015a] Goroshin, R., Bruna, J., Tompson, J., Eigen, D., and LeCun, Y. (2015a). Unsupervised learning of spatiotemporally coherent metrics. *International Conference on Conputer Vision (ICCV)*.

[Goroshin et al., 2015b] Goroshin, R., Mathieu, M., and LeCun, Y. (2015b). Learning to linearize under uncertainty. *NIPS*.

[Hannun et al., 2014] Hannun, A. Y., Case, C., Casper, J., Catanzaro, B., Diamos, G., Elsen, E., Prenger, R., Satheesh, S., Sengupta, S., Coates, A., and Ng, A. Y. (2014). Deep speech: Scaling up end-to-end speech recognition. *CoRR*, abs/1412.5567.

[He et al., 2015] He, K., Zhang, X., Ren, S., and Sun, J. (2015). Deep residual learning for image recognition. *CoRR*, abs/1512.03385.

[Hinton et al., 2011] Hinton, G. E., Krizhevsky, A., and Wang, S. D. (2011). Transforming auto-encoders. In *Proceedings of the 21st International Conference on Artificial Neural Networks - Volume Part I*, ICANN'11, pages 44–51, Berlin, Heidelberg. Springer-Verlag.

[Hochreiter and Schmidhuber, 1997] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Comput.*, 9(8):1735–1780.

[Hyvärinen et al., 2004] Hyvärinen, Aapo, Karhunen, Juha, Oja, and Erkki (2004). *Independent component analysis*, volume 46. John Wiley & Sons.

[Im et al., 2016] Im, D. J., Kim, C. D., Jiang, H., and Memisevic, R. (2016). Generating images with recurrent adversarial networks. *arXiv preprint arXiv:1602.05110*.

[Ioffe and Szegedy, 2015] Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*.

[Jain et al., 2007] Jain, V., Murray, J. F., Roth, F., Turaga, S., Zhigulin, V., Briggman, K. L., N, H. M., Denk, W., and Seung, S. H. (2007). Supervised learning of image restoration with convolutional networks.

[Jayaraman and Grauman, 2015] Jayaraman, D. and Grauman, K. (2015). Learning image representations equivariant to ego-motion. In *ICCV*.

[Karpathy and Li, 2014] Karpathy, A. and Li, F. (2014). Deep visual-semantic alignments for generating image descriptions. *CoRR*, abs/1412.2306.

[Karpathy et al., 2014] Karpathy, A., Toderici, G., Shetty, S., Leung, T., Sukthankar, R., and Fei-Fei, L. (2014). Large-scale video classification with convolutional neural networks. In *CVPR*.

[Kavukcuoglu et al., 2010a] Kavukcuoglu, K., Ranzato, M., and LeCun, Y. (2010a). Fast inference in sparse coding algorithms with applications to object recognition. *CoRR*, abs/1010.3467.

[Kavukcuoglu et al., 2010b] Kavukcuoglu, K., Sermanet, P., Boureau, Y.-L., Gregor, K., Mathieu, M., and LeCun, Y. (2010b). Learning convolutional feature hierarchies for visual recognition. In *NIPS*.

[Kayser et al., 2001] Kayser, C., Einhauser, W., Dummer, O., Konig, P., and Kding, K. (2001). Extracting slow subspaces from natural videos leads to complex cells. In *ICANN'2001*.

[Kim and Bengio, 2016] Kim, T. and Bengio, Y. (2016). Deep directed generative models with energy-based probability estimation. *CoRR*, abs/1606.03439.

[Kingma and Ba, 2014] Kingma, D. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

[Kingma et al., 2014] Kingma, D. P., Mohamed, S., Rezende, D. J., and Welling, M. (2014). Semi-supervised learning with deep generative models. In *Advances in Neural Information Processing Systems*, pages 3581–3589.

[Kingma and Welling, 2013] Kingma, D. P. and Welling, M. (2013). Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.

[Kosaka and Kak, 1992] Kosaka, A. and Kak, A. C. (1992). Fast vision-guided mobile robot navigation using model-based reasoning and prediction of uncertainties. *CVGIP: Image understanding*, 56(3):271–329.

[Krishnan et al., 2011] Krishnan, D., Tay, T., and Fergus, R. (2011). Blind deconvolution using a normalized sparsity measure. In *CVPR*, pages 233–240.

[Krizhevsky et al., 2012] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *NIPS*, volume 1, page 4.

[Kulkarni et al., 2015] Kulkarni, T. D., Whitney, W. F., Kohli, P., and Tenenbaum, J. (2015). Deep convolutional inverse graphics network. In *Advances in Neural Information Processing Systems*, pages 2530–2538.

[Larsen et al., 2015] Larsen, A. B. L., Sønderby, S. K., and Winther, O. (2015). Autoencoding beyond pixels using a learned similarity metric. *arXiv preprint arXiv:1512.09300*.

[Le, 2013] Le, Q. V. (2013). Building high-level features using large scale unsupervised learning. In *ICASSP*, pages 8595–8598.

[LeCun et al., 2015] LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature*, 521(7553):436–444. Insight.

[LeCun et al., 1998] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proc. IEEE*, 86(11):2278–2324.

[Lecun et al., 2006] Lecun, Y., Chopra, S., Hadsell, R., Huang, F. J., and Ranzato, M. A. (2006). *A Tutorial on Energy-Based Learning*. MIT Press.

[LeCun et al., 2004] LeCun, Y., Huang, F. J., and Bottou, L. (2004). Learning methods for generic object recognition with invariance to pose and lighting. In *CVPR*.

[Ledig et al., 2016] Ledig, C., Theis, L., Huszar, F., Caballero, J., Aitken, A. P., Tejani, A., Totz, J., Wang, Z., and Shi, W. (2016). Photo-realistic single image super-resolution using a generative adversarial network. *CoRR*, abs/1609.04802.

[Lin et al., 2014] Lin, T., Maire, M., Belongie, S. J., Bourdev, L. D., Girshick, R. B., Hays, J., Perona, P., Ramanan, D., Dollár, P., and Zitnick, C. L. (2014). Microsoft COCO: common objects in context. *CoRR*, abs/1405.0312.

[Liu et al., 2015] Liu, Z., Luo, P., Wang, X., and Tang, X. (2015). Deep learning face attributes in the wild. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3730–3738.

[Lloyd, 1982] Lloyd, S. (1982). Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 28(2):129–137.

[Long et al., 2015] Long, J., Shelhamer, E., and Darrell, T. (2015). Fully convolutional networks for semantic segmentation. In *CVPR*.

[Louizos et al., 2016] Louizos, C., Swersky, K., Li, Y., Welling, M., and Zemel, R. (2016). The variational fair autoencoder. *ICLR*.

[Luc et al., 2016] Luc, P., Couprie, C., Chintala, S., and Verbeek, J. (2016). Semantic segmentation using adversarial networks. *CoRR*, abs/1611.08408.

[Mahendran and Vedaldi, 2015] Mahendran, A. and Vedaldi, A. (2015). Understanding deep image representations by inverting them. In *CVPR*.

[Makhzani and Frey, 2013] Makhzani, A. and Frey, B. J. (2013). k-sparse autoencoders. *CoRR*, abs/1312.5663.

[Makhzani and Frey, 2014] Makhzani, A. and Frey, B. J. (2014). A winner-take-all method for training sparse convolutional autoencoders. *CoRR*, abs/1409.2752.

[Makhzani et al., 2015] Makhzani, A., Shlens, J., Jaitly, N., and Goodfellow, I. J. (2015). Adversarial autoencoders. *CoRR*, abs/1511.05644.

[Mathieu et al., 2015] Mathieu, M., Couprie, C., and LeCun, Y. (2015). Deep multi-scale video prediction beyond mean square error. *ICLR*, abs/1511.05440.

[Mathieu et al., 2016] Mathieu, M., Zhao, J. J., Sprechmann, P., Ramesh, A., and LeCun, Y. (2016). Disentangling factors of variation in deep representations using adversarial training. *CoRR*, abs/1611.03383.

[Mescheder et al., 2017] Mescheder, L. M., Nowozin, S., and Geiger, A. (2017). Adversarial variational bayes: Unifying variational autoencoders and generative adversarial networks. *CoRR*, abs/1701.04722.

[Metz et al., 2016] Metz, L., Poole, B., Pfau, D., and Sohl-Dickstein, J. (2016). Unrolled generative adversarial networks. *CoRR*, abs/1611.02163.

[Mikolov, 2010] Mikolov, T. (2010). Recurrent neural network based language model.

[Mirza and Osindero, 2014] Mirza, M. and Osindero, S. (2014). Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*.

[Mnih et al., 2013] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. A. (2013). Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602.

[Mobahi et al., 2009] Mobahi, H., Collobert, R., and Weston, J. (2009). Deep learning from temporal coherence in video. In *ICML*.

[Nair and Hinton, 2010] Nair, V. and Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. In *ICML*, pages 807–814.

[Narayanan and Mitter, 2010] Narayanan, H. and Mitter, S. (2010). Sample complexity of testing the manifold hypothesis. In *Advances in Neural Information Processing Systems*, pages 1786–1794.

[Nathan Silberman and Fergus, 2012] Nathan Silberman, Derek Hoiem, P. K. and Fergus, R. (2012). Indoor segmentation and support inference from RGBD images. In *ECCV*.

[Nowozin et al., 2016] Nowozin, S., Cseke, B., and Tomioka, R. (2016). f-gan: Training generative neural samplers using variational divergence minimization. *CoRR*.

[Oh et al., 2015] Oh, J., Guo, X., Lee, H., Lewis, R. L., and Singh, S. P. (2015). Action-conditional video prediction using deep networks in atari games. *NIPS*.

[Olshausen and Field, 1997] Olshausen, B. A. and Field, D. J. (1997). Sparse coding with an overcomplete basis set: A strategy employed by v1? *Vision research*, 37(23):3311–3325.

[Pearson, 1901] Pearson, K. (1901). On lines and planes of closest fit to system of points in space. philosophical magazine, 2, 559-572.

[Pezeshki et al., 2015] Pezeshki, M., Fan, L., Brakel, P., Courville, A., and Bengio, Y. (2015). Deconstructing the ladder network architecture. *arXiv preprint arXiv:1511.06430*.

[Poultney et al., 2006] Poultney, C., Chopra, S., Cun, Y. L., et al. (2006). Efficient learning of sparse representations with an energy-based model. In *Advances in neural information processing systems*, pages 1137–1144.

[Radford et al., 2015] Radford, A., Metz, L., and Chintala, S. (2015). Unsupervised representation learning with deep convolutional generative adversarial networks. *CoRR*, abs/1511.06434.

[Ranzato et al., 2007a] Ranzato, M., Boureau, Y., Chopra, S., and LeCun, Y. (2007a). A unified energy-based framework for unsupervised learning. In *Proc. Conference on AI and Statistics (AI-Stats)*.

[Ranzato et al., 2007b] Ranzato, M., Boureau, Y.-L., and LeCun, Y. (2007b). Sparse feature learning for deep belief networks. *Advances in neural information processing systems*, 20:1185–1192.

[Ranzato et al., 2014] Ranzato, M., Szlam, A., Bruna, J., Mathieu, M., Collobert, R., and Chopra, S. (2014). Video (language) modeling: a baseline for generative models of natural videos. *CoRR*, abs/1412.6604.

[Ranzato et al., 2007c] Ranzato, M. A., Huang, F. J., Boureau, Y.-L., and LeCun, Y. (2007c). Unsupervised learning of invariant feature hierarchies with applications to object recognition. In *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*, pages 1–8. IEEE.

[Rasmus et al., 2015] Rasmus, A., Berglund, M., Honkala, M., Valpola, H., and Raiko, T. (2015). Semi-supervised learning with ladder networks. In *Advances in Neural Information Processing Systems*, pages 3546–3554.

[Razavian et al., 2014] Razavian, A. S., Azizpour, H., Sullivan, J., and Carlsson, S. (2014). CNN features off-the-shelf: an astounding baseline for recognition. *CoRR*, abs/1403.6382.

[Reed et al., 2014] Reed, S., Sohn, K., Zhang, Y., and Lee, H. (2014). Learning to disentangle factors of variation with manifold interaction. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 1431–1439.

[Reed et al., 2015] Reed, S. E., Zhang, Y., Zhang, Y., and Lee, H. (2015). Deep visual analogy-making. In *Advances in Neural Information Processing Systems 28*, pages 1252–1260. Curran Associates, Inc.

[Revaud et al., 2015] Revaud, J., Weinzaepfel, P., Harchaoui, Z., and Schmid, C. (2015). EpicFlow: Edge-Preserving Interpolation of Correspondences for Optical Flow. In *Computer Vision and Pattern Recognition*.

[Rezende et al., 2014] Rezende, D. J., Mohamed, S., and Wierstra, D. (2014). Stochastic backpropagation and approximate inference in deep generative models. *arXiv preprint arXiv:1401.4082*.

[Rifai et al., 2011] Rifai, S., Vincent, P., Muller, X., Glorot, X., and Bengio, Y. (2011). Contractive auto-encoders: Explicit invariance during feature extraction. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 833–840.

[Ronneberger et al., 2015] Ronneberger, O., Fischer, P., and Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. In *MICCAI*, volume 9351, pages 234–241.

[Russakovsky et al., 2015] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. (2015). ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252.

[Salimans et al., 2016] Salimans, T., Goodfellow, I. J., Zaremba, W., Cheung, V., Radford, A., and Chen, X. (2016). Improved techniques for training gans. *CoRR*, abs/1606.03498.

[Sermanet et al., 2013] Sermanet, P., Eigen, D., Zhang, X., Mathieu, M., Fergus, R., and LeCun, Y. (2013). Overfeat: Integrated recognition, localization and detection using convolutional networks. *CoRR*, abs/1312.6229.

[Silver et al., 2016] Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. (2016). Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489.

[Simonyan and Zisserman, 2014] Simonyan, K. and Zisserman, A. (2014). Two-stream convolutional networks for action recognition in videos. *CoRR*, abs/1406.2199.

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

[Soomro et al., 2012] Soomro, K., Zamir, A. R., and Shah, M. (2012). UCF101: A dataset of 101 human actions classes from videos in the wild. *CoRR*, abs/1212.0402.

[Srivastava et al., 2015] Srivastava, N., Mansimov, E., and Salakhutdinov, R. (2015). Unsupervised learning of video representations using LSTMs. In *ICML*.

[Sutskever et al., 2014] Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. *CoRR*, abs/1409.3215.

[Szegedy et al., 2013] Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I. J., and Fergus, R. (2013). Intriguing properties of neural networks. *CoRR*, abs/1312.6199.

[Tenenbaum and Freeman, 2000] Tenenbaum, J. B. and Freeman, W. T. (2000). Separating style and content with bilinear models. *Neural Comput.*, 12(6):1247–1283.

[Theis et al., 2015] Theis, L., Oord, A. v. d., and Bethge, M. (2015). A note on the evaluation of generative models. *arXiv preprint arXiv:1511.01844*.

[Tran et al., 2015] Tran, D., Bourdev, L. D., Fergus, R., Torresani, L., and Paluri, M. (2015). C3D: generic features for video analysis. In *ICCV*.

[Uria et al., 2016] Uria, B., Côté, M., Gregor, K., Murray, I., and Larochelle, H. (2016). Neural autoregressive distribution estimation. *CoRR*, abs/1605.02226.

[van den Oord et al., 2016a] van den Oord, A., Kalchbrenner, N., and Kavukcuoglu, K. (2016a). Pixel recurrent neural networks. *CoRR*, abs/1601.06759.

[van den Oord et al., 2016b] van den Oord, A., Kalchbrenner, N., Vinyals, O., Espeholt, L., Graves, A., and Kavukcuoglu, K. (2016b). Conditional image generation with pixelcnn decoders. *CoRR*, abs/1606.05328.

[Villegas et al., 2017] Villegas, R., Yang, J., Zou, Y., Sohn, S., Lin, X., and Lee, H. (2017). Learning to generate long-term future via hierarchical prediction. arXiv:1704.05831.

[Vincent et al., 2008] Vincent, P., Larochelle, H., Bengio, Y., and Manzagol, P.-A. (2008). Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103. ACM.

[Vincent et al., 2010] Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., and Manzagol, P.-A. (2010). Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of Machine Learning Research*, 11(Dec):3371–3408.

[Vondrick et al., 2016a] Vondrick, C., Pirsiavash, H., and Torralba, A. (2016a). Anticipating visual representations from unlabeled video. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[Vondrick et al., 2016b] Vondrick, C., Pirsiavash, H., and Torralba, A. (2016b). Generating videos with scene dynamics. In *Advances In Neural Information Processing Systems*, pages 613–621.

[Wang and Gupta, 2015] Wang, X. and Gupta, A. (2015). Unsupervised learning of visual representations using videos. In *ICCV*.

[Wang et al., 2004] Wang, Z., Bovik, A. C., Sheikh, H. R., and Simoncelli, E. P. (2004). Image quality assessment: From error visibility to structural similarity. *IEEE Trans. on Im. Proc.*, 13(4):600–612.

[Weinzaepfel et al., 2013] Weinzaepfel, P., Revaud, J., Harchaoui, Z., and Schmid, C. (2013). Deepflow: Large displacement optical flow with deep matching. In *2013 IEEE International Conference on Computer Vision*, pages 1385–1392.

[Wiskott and Sejnowski, 2002] Wiskott, L. and Sejnowski, T. J. (2002). Slow feature analysis: Unsupervised learning of invariances. *Neural Computation*.

[Wolpert, 1996] Wolpert, D. H. (1996). The lack of a priori distinctions between learning algorithms. *Neural Computation*, 8(7):1341–1390.

[Yu et al., 2015] Yu, F., Seff, A., Zhang, Y., Song, S., Funkhouser, T., and Xiao, J. (2015). Lsun: Construction of a large-scale image dataset using deep learning with humans in the loop. *arXiv preprint arXiv:1506.03365*.

[Zeiler et al., 2010] Zeiler, M. D., Krishnan, D., Taylor, G. W., and Fergus, R. (2010). Deconvolutional networks. In *Computer Vision and Pattern Recognition (CVPR)*, pages 2528–2535.

[Zemel, 1994] Zemel, R. S. (1994). Autoencoders, minimum description length and Helmholtz free energy.

[Zen and Senior, 2014] Zen, H. and Senior, A. (2014). Deep mixture density networks for acoustic modeling in statistical parametric speech synthesis. In *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3844–3848.

[Zhao et al., 2016a] Zhao, J., Mathieu, M., Goroshin, R., and LeCun, Y. (2016a). Stacked what-where auto-encoders. In *ICLR workshop submission*.

[Zhao et al., 2016b] Zhao, J. J., Mathieu, M., and LeCun, Y. (2016b). Energy-based generative adversarial network. *CoRR*, abs/1609.03126.