Université de Montréal

Representation Learning for Visual Data

par Vincent Dumoulin

Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences

Thèse présentée à la Faculté des arts et des sciences
en vue de l'obtention du grade de Philosophiæ Doctor (Ph.D.)
en informatique

Septembre, 2018

Résumé

Cette thèse par article contribue au domaine de l'apprentissage de représentations profondes, et plus précisement celui des modèles génératifs profonds, par l'entremise de travaux sur les machines de Boltzmann restreintes, les modèles génératifs adversariels ainsi que le pastiche automatique.

Le premier article s'intéresse au problème de l'estimation du gradient de la phase négative des machines de Boltzmann par l'échantillonnage d'une réalisation physique du modèle. Nous présentons une évaluation empirique de l'impact sur la performance, mesurée par log-vraisemblance négative, de diverses contraintes associées à l'implémentation physique de machines de Boltzmann restreintes (RBMs), soit le bruit sur les paramètres, l'amplitude limitée des paramètres et une connectivité limitée.

Le second article s'attaque au problème de l'inférence dans les modèles génératifs adversariels (GANs). Nous proposons une extension du modèle appelée inférence adversativement apprise (ALI) qui a la particularité d'apprendre jointement l'inférence et la génération à partir d'un principe adversariel. Nous montrons que la représentation apprise par le modèle est utile à la résolution de tâches auxiliaires comme l'apprentissage semi-supervisé en obtenant une performance comparable à l'état de l'art pour les ensembles de données SVHN et CIFAR10.

Finalement, le troisième article propose une approche simple et peu coûteuse pour entraîner un réseau unique de pastiche automatique à imiter plusieurs styles artistiques. Nous présentons un mécanisme de conditionnement, appelé normalisation conditionnelle par instance, qui permet au réseau d'imiter plusieurs styles en parallèle via l'apprentissage d'un ensemble de paramètres de normalisation unique à chaque style. Ce mécanisme s'avère très efficace en pratique et a inspiré plusieurs travaux subséquents qui ont appliqué l'idée à des problèmes au-delà du domaine du pastiche automatique.

Mots-clés: réseaux neuronaux, apprentissage automatique, apprentissage de représentations profondes, apprentissage non supervisé, modèles à énergie, calcul par système physique, modèles génératifs, réseaux adversariels génératifs, synthèse d'image, pastiche automatique

Summary

This thesis by articles contributes to the field of deep learning, and more specifically the subfield of deep generative modeling, through work on restricted Boltzmann machines, generative adversarial networks and style transfer networks.

The first article examines the idea of tackling the problem of estimating the negative phase gradients in Boltzmann machines by sampling from a physical implementation of the model. We provide an empirical evaluation of the impact of various constraints associated with physical implementations of restricted Boltzmann machines (RBMs), namely noisy parameters, finite parameter amplitude and restricted connectivity patterns, on their performance as measured by negative log-likelihood through software simulation.

The second article tackles the inference problem in generative adversarial networks (GANs). It proposes a simple and straightforward extension to the GAN framework, named adversarially learned inference (ALI), which allows inference to be learned jointly with generation in a fully-adversarial framework. We show that the learned representation is useful for auxiliary tasks such as semi-supervised learning by obtaining a performance competitive with the then-state-of-the-art on the SVHN and CIFAR10 semi-supervised learning tasks.

Finally, the third article proposes a simple and scalable technique to train a single feedforward style transfer network to model multiple styles. It introduces a conditioning mechanism named conditional instance normalization which allows the network to capture multiple styles in parallel by learning a different set of instance normalization parameters for each style. This mechanism is shown to be very efficient and effective in practice, and has inspired multiple efforts to adapt the idea to problems outside of the artistic style transfer domain.

Keywords: neural network, machine learning, deep learning, unsupervised learning, energy-based models, physical computing, generative modeling, generative adversarial network, image synthesis, style transfer

Contents

List of Figures

List of Tables

List of Abbreviations

AI	Artificial intelligence
AIS	Annealed importance sampling
ALI	Adversarially learned inference
BiGAN	Bidirectional GAN
CD	Contrastive divergence
CNN	Convolutional neural network
CVAE	Convolutional variational autoencoder
DRAW	Deep recurrent attentive writer
ELBO	Evidence lower-bound
EM	Expectation-maximization
GAN	Generative adversarial network
GPU	Graphical processing unit
KL	Kullback-Leibler, as in "Kullback-Leibler divergence"
LAPGAN	Laplacian pyramid of generative adversarial networks
NLL	Negative log-likelihood
PCD	Persistent contrastive divergence
PGM	Probabilistic graphical model
RBM	Restricted Boltzmann machine
RNN	Recurrent neural network
SGD	Stochastic gradient descent
VAE	Variational autoencoder

Acknowledgments

I would like to thank my mother Marie-Hélène Labrecque and my father Yves Dumoulin for their unconditional support during the course of my studies. From as far as I can recall, they encouraged me to pursue my intellectual curiosity, which I think is one of the greatest gifts one can give to a child.

My mother is an example of kindness, empathy and resilience which I try to emulate as best as I can. Completing a PhD is an emotional ride with its ups and downs, and she, along with my sister Cassandra Dumoulin and my partner Anne-Sophie Gendron, was a comforting and invaluable presence during my times of doubt.

I retain from my father the will to be fully invested in my passions which he always encouraged me to cultivate. He was and remains my best audience when comes the time to share my excitement about what interests me and my victories, small and big.

I would like to thank my partner Anne-Sophie Gendron for her love and support, and for being patient and understanding during the inevitable rush periods preceding submission deadlines. I would also like to thank Claude Roy for sparking my interest in research and in pursuing a graduate degree.

My first contact with machine learning in general, and deep learning in particular, was from a talk my advisor, Yoshua Bengio, gave in the physics department at Université de Montréal. Were it not for his contagious passion and his willingness to take me as an undergraduate summer intern in his lab, I would not be where I stand today. His commitment to understanding intelligence and to fostering a vibrant research community is an incredible source of inspiration for me. I would also like to thank my co-advisor, Aaron Courville, for the countless discussions which we shared over the years and which helped shape my view of deep learning. His openness to hearing other points of view and being challenged on his ideas gave me the confidence to express and defend mine freely and the humility to welcome the ones of other people.

Finally, I would like to thank the many students whose paths I crossed during my PhD. In particular, I would like to thank Ian Goodfellow and David Warde-Farley for taking the time to help a then-junior graduate student with both theory and practice and to patiently answer the many questions he had. Deep learning research is as much about having good ideas as it is to implement them and share them with the world, and their help has proven invaluable for the latter.

1 Background

Artificial intelligence (AI) is a subject which captivates a lot of people. The long-term goal of AI is to emulate biological intelligence, which is a very hard problem to solve. We will refer to this as *general AI*. In the short term, AI's objective is to solve tasks without the need for human intervention. This is what we call *specific AI*.

The "classical" approach to solving specific AI problems is to translate human knowledge into instructions that a computer can understand. For instance, a program can be written to control the speed of a vehicule using simple rules: if the vehicule's speed falls below 60 km/h, start accelerating, and if its speed goes above 100 km/h, stop accelerating.

1.1 Machine learning

However, in many cases, this traditional approach fails, because human knowledge is not always easily put into words. Oftentimes, human experts rely on intuitions that are gained through experience without ever being explicitly stated. For instance, the act of walking does not require an explicit cognitive process, and most people are unaware of the small adjustments their body performs to keep them in balance. Even though recognizing a face is something we do naturally, it is very difficult to describe what a face is without using abstract concepts, such as *eyes*, *nose* and *mouth*, which themselves rely on abstract concepts to be described.

Instead of relying on prior knowledge and expertise to solve specific AI tasks, another approach would be to let the program learn from data. This is what we call *machine learning*.

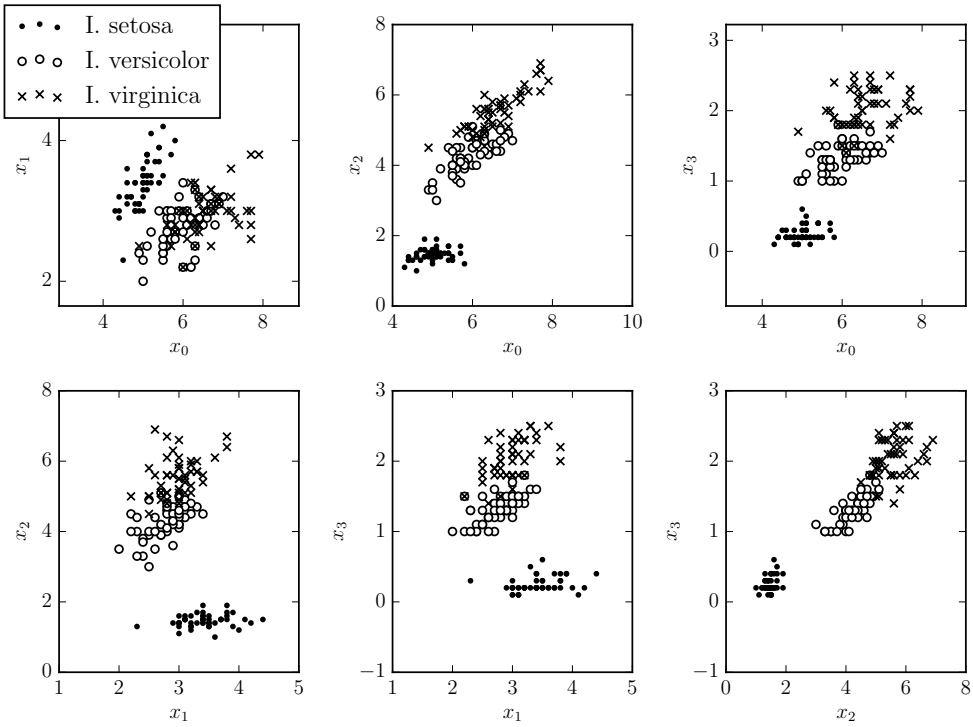


Figure 1.1 – Pairwise scatter plots of the Iris features.

1.1.1 Formalism

Datasets and data-generating distributions

In order to introduce machine learning algorithms more formally, let us consider the concrete example of the Iris dataset (Fisher, 1936), which is a set of 150 observations of iris flowers of 3 different species. Each observation is composed of 4 real-valued *features*, each of which quantifies a different flower characteristic, as well as a categorical feature corresponding to the flower species (Figure 1.1).

The set of all observations is noted

$$\mathcal{D} = \{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(N)}\}, \tag{1.1}$$

where $\mathbf{x}^{(i)} = (x_1^{(i)}, \dots, x_D^{(i)})^T$ is a single observation. In our example, $N = 150$ is the number of observations and the semantics of $\mathbf{x} \in \mathcal{D}$ are outlined in Table 1.1.

Note that $\mathcal{D}$ is only a subset of all observations that can be made: there are certainly more than 150 iris flowers in the world, and if we were to make another set

x_i	Domain	Description
x_1	$\mathbb{R}$	Sepal length
x_2	$\mathbb{R}$	Sepal width
x_3	$\mathbb{R}$	Petal length
x_4	$\mathbb{R}$	Petal width
x_5	$\{0, 1, 2\}$	Species

Table 1.1 – Semantics of an Iris observation.

of 150 observations, we would be getting different results. In fact, all observations come from some *data-generating distribution* $P(\mathbf{x})$ that is unknown to us.

Parametric and non-parametric functions

In order to do something useful with these observations, we need to have a function that receives an input and produces an output.

This function can either be *parametric* or *non-parametric*. Parametric functions are described by a fixed number of parameters, whereas non-parametric functions have a number of parameters that grows with $|\mathcal{D}|$ (e.g. the nearest neighbours algorithm).

We will concentrate on families of parametric functions, noted $\mathcal{F}$. Each function $f \in \mathcal{F}$ corresponds to a specific assignment of a set of parameters θ. For instance, we could choose $\mathcal{F}$ to be the family of all linear functions mapping the first three real-valued features of Iris to some real number y:

$$\mathcal{F} = \{f_\theta : \mathbb{R}^3 \to \mathbb{R} \mid y = f_\theta(\mathbf{x}_{1:3}) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3\} \tag{1.2}$$

Different values of θ correspond to different functions in $\mathcal{F}$. For example,

$$\begin{aligned} \theta = (0, 1, 1, 1) &\Rightarrow y = x_1 + x_2 + x_3, \\ \theta = (1, -1, 1, -1) &\Rightarrow y = 1 - x_1 + x_2 - x_3. \end{aligned} \tag{1.3}$$

Loss function

The choice of θ is guided by the task we wish to solve. Ideally, θ would be such that f_θ performs well on all observations $\mathbf{x} \sim P(\mathbf{x})$. Unfortunately, since we do not have access to the data-generating distribution, the best we can do is to concentrate on $\mathcal{D}$.

For this, we introduce a *loss function* $\mathcal{L}(\theta, \mathcal{D})$, which quantifies how bad f_θ performs on our task for all $\mathbf{x} \in \mathcal{D}$ (higher values are worse). We wish to find a value $\theta_{\min}$ that minimizes $\mathcal{L}$, that is,

$$\theta_{\min} = \arg\min_{\theta} \mathcal{L}(\theta, \mathcal{D}). \tag{1.4}$$

In other words, assuming some "true" underlying function f^* which we are trying to approximate, $\mathcal{L}$ measures how badly f_θ "deviates" from f^*.

Going forward with Equation 1.2, let us pretend that we want to predict the value of x_4 based on $\mathbf{x}_{1:3}$. One choice of $\mathcal{L}$ could be

$$\mathcal{L}(\theta, \mathcal{D}) = \sum_{\mathbf{x} \in \mathcal{D}} (f_\theta(\mathbf{x}_{1:3}) - x_4)^2, \tag{1.5}$$

i.e., we consider $f_\theta(\mathbf{x}_{1:3})$ to be a linear predictor of x_4, and we wish to penalize large discrepancies between the predicted value $f_\theta(\mathbf{x}_{1:3})$ and the true value x_4. We would then set out to find the value $\theta_{\min}$ that minimizes $\mathcal{L}(\theta, \mathcal{D})$ for our specific problem instance.

Generalization and the bias-variance trade-off

Note that finding a $\theta_{\min}$ that minimizes $\mathcal{L}$ on observations from the finite dataset $\mathcal{D}$ sampled from $P(\mathbf{x})$ does not guarantee that $f_{\theta_{\min}}$ will *generalize* well, i.e. it will perform well on unseen observations.

On one hand, if $\mathcal{F}$ is not rich enough (e.g., if $\mathcal{F}$ is the family of linear functions while f^* is a quadratic function), then $f_{\theta_{\min}}$ will deviate from f^* in expectation over datasets $\mathcal{D}$ sampled from $P(\mathbf{x})$ — in other words, $f_{\theta_{\min}}$ will be a biased estimate of f^*.

On the other hand, if $\mathcal{F}$ is rich enough to reduce $\mathcal{L}(\theta_{\min}, \mathcal{D})$ to zero for any dataset, the function $f_{\theta_{\min}}$ obtained through learning may be very sensitive to the specific values of observations in $\mathcal{D}$. The function will suffer from high variance, i.e.

it will on average deviate largely from its expected value over datasets $\mathcal{D}$ sampled from $P(\mathbf{x})$.

Model selection

In choosing $\mathcal{F}$, we have to trade between bias and variance. The correct trade-off is impossible to determine in advance. Instead, we can select a subset of $\mathcal{D}$, called the *test set*, which will not be used to select $\theta_{\min}$. We will use the test set to obtain an unbiased estimate of how well $f_{\theta_{\min}}$ generalizes (in the sense that the difference between the loss measured on the test set and the loss measured on the whole data-generating distribution is zero in expectation). The family $\mathcal{F}$ with the best bias-variance trade-off is the one that has the best generalization performance.

However, in choosing $\mathcal{F}$, we have introduced a bias in our generalization performance estimate: $\mathcal{F}$ performs well on the test set *because it was chosen for that very reason*, which means that the difference between the loss measured on the test set and the loss measured on the whole data-generating distribution is no longer zero in expectation. This is why in practice we often split $\mathcal{D}$ into three sets:

1. the *training set*, which is used to find $\theta_{\min}$,
2. the *validation set*, which is used to choose $\mathcal{F}$, and
3. the *test set*, which is used to obtain an unbiased estimate of the model's generalization performance.

1.1.2 Machine learning problems

The problems machine learning attempts to solve fall into three broad categories:

1. supervised learning,
2. unsupervised learning, and
3. reinforcement learning.

Supervised learning

A problem for which both the input and the expected output are given is called a *supervised learning* problem. We are interested in predicting

1. categorical outputs (*classification*) or
2. real-valued outputs (*regression*).

In our Iris example, predicting the species of an observation (x_5) given its real-valued features ($\mathbf{x}_{1:4}$) would be an instance of a classification problem, whereas predicting x_4 given $\mathbf{x}_{1:3}$ would correspond to a regression problem.

Unsupervised learning

An *unsupervised learning* problem is a problem for which there is no specific output to predict. Instead, we are interested in discovering underlying structure in the inputs we are observing, such as

1. grouping examples together (*clustering*),
2. finding a mathematical description of where examples are likely to be found in feature space (*density estimation*), or
3. finding a more compact representation of the examples that preserves useful information (*dimensionality reduction*).

In our Iris example, trying to group observations together based only on $\mathbf{x}_{1:4}$ is an instance of a clustering problem. Trying to predict whether a $\mathbf{x}_{1:4}$ configuration corresponds to a plausible iris flower observation would correspond to a density estimation problem. Finally, finding a 2D representation of the observations that preserves most of the information in order to display Figure 1.1 in a single scatter plot would be an instance of a dimensionality reduction problem.

Reinforcement learning

Reinforcement learning is somewhat different than supervised and unsupervised learning. We will briefly describe it for the sake of completion.

In reinforcement learning, an *agent* is allowed to interact with a stateful *environment*. Certain *states* are rewarded, while other states are penalized. The *actions* the agent takes affect the state of the environment. The agent is trained to find a *policy*, mapping states to actions, which maximizes the reward it receives.

For instance, an autonomous helicopter (the agent) could be trained to stay in flight as long as it can. Each time frame spent in the air is rewarded, and crashes are penalized. The helicopter would learn a policy mapping its state (e.g. linear

speed, angular speed, linear acceleration, angular acceleration, altitude) to actions (e.g. adjusting the pitch of rotors) that allow it to stay airborne.

1.1.3 Training

Training, or *learning*, is the act of finding a $\theta_{\min}$ that minimizes $\mathcal{L}(\theta, \mathcal{D})$. Although some machine learning algorithms (like linear regression) allow an analytical solution, most algorithms rely on *gradient-based optimization* for training.

Gradient descent

If $\mathcal{L}(\theta, \mathcal{D})$ is differentiable with respect to θ, we can iteratively reduce $\mathcal{L}$ with the following parameter update equation:

$$\theta \leftarrow \theta - \alpha \frac{\partial}{\partial \theta} \mathcal{L}(\theta, \mathcal{D}). \tag{1.6}$$

This is what we call *gradient descent*. The α scalar is called the *learning rate* and controls the pace at which θ travels along the loss function's gradient.

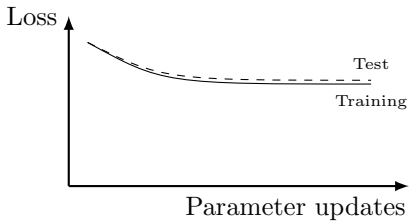
For large datasets, computing the gradient for all observations may be very expensive. An alternative is to update parameters using one or a few examples at a time. This called *stochastic gradient descent* (SGD), because the gradient with respect to one observation is a stochastic approximation of the true gradient (assuming that our observations are coming from the same distribution). Even though the gradient is noisy, this approach benefits from a faster convergence rate (Bousquet and Bottou, 2008).

As a middle ground, people often work with small batches of data (called *mini-batches*) at a time to leverage the parallelization offered by computing hardware, such as graphical processing units (GPUs).

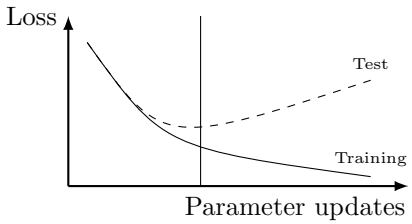
Learning diagnostics

One of the best tools available to troubleshoot learning is the *learning curve* (Figure 1.2), a plot of the training and test losses as a function of the number of parameter updates.

When we fail to obtain a sufficiently low training loss, we are in an *underfitting* regime (Figure 1.2a). This is related to the bias problem: if we make a poor choice



(a) This profile is common in underfitting settings.



(b) Overfitting regime, after the vertical bar is passed.

Figure 1.2 – Learning curves.

of $\mathcal{F}$, no learning can make the model perform very well. Note that other issues may also cause underfitting, such as the presence of local minima or saddlepoints (for non-convex loss functions) or a high variance on the gradient (when the gradient is computed using stochastic approaches).

When there is a significant gap between the training and test losses, we are in an *overfitting* regime (Figure 1.2b): we have failed to generalize to unseen examples. This is related to the variance problem: the model adapts to accidental variations in the dataset which would not be there if we had access to the whole data-generating distribution, and it loses generalization power.

Preventing overfitting may be done using various approaches:

— Choosing a less flexible $\mathcal{F}$.

— Adding a *regularization* penalty to the loss function that induces a preference over the functions of $\mathcal{F}$.

— Monitoring the validation loss and choosing the θ associated with the lowest validation loss (*early stopping*).

1.1.4 Artificial neural networks

One very popular type of function families is what we call *artificial neural networks*. The best way to describe what they are is to show how the output of one such function is computed.

We start with the input $\mathbf{x}$, which is called the *visible layer*. We typically apply an affine transformation to it followed by an elementwise nonlinearity:

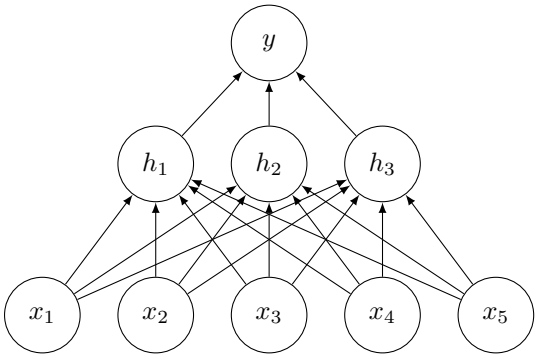


Figure 1.3 – Neural network with one hidden layer.

$$\begin{aligned} \mathbf{a}_1 &= W_1\mathbf{x} + \mathbf{b}_1, \\ h_{1,i} &= g(a_{1,i}) \end{aligned} \tag{1.7}$$

The resulting vector $\mathbf{h}_1$ is called the first *hidden layer*. We distinguish between the *pre-activations* $\mathbf{a}_1$ and the *activations* $\mathbf{h}_1$. Nonlinearities often used in practice include

— the *logistic function* $\sigma(x) = (1 + \exp(-x))^{-1}$,

— the *hyperbolic tangent function*, and

— the *rectified linear function* $\mathrm{ReLU}(x) = \max(0, x)$.

This process is iteratively repeated for all M hidden layers of the neural network:

$$\begin{aligned} \mathbf{a}_m &= W_m\mathbf{h}_{m-1} + \mathbf{b}_m, \\ h_{m,i} &= g(a_{m,i}) \end{aligned} \tag{1.8}$$

The last layer of a neural network, called the *output layer*, usually differs from other layers and its form depends on the type of problem we are trying to solve. If we are doing regression, the output layer might look like

$$y = \mathbf{w}_{\text{out}}^T\mathbf{h}_M + \mathbf{b}_{\text{out}} \tag{1.9}$$

whereas for classification problems we would use

$$\begin{aligned} \mathbf{a}_{\text{out}} &= \mathbf{W}_{\text{out}}^T \mathbf{x}_M + \mathbf{b}_{\text{out}}, \\ \mathbf{y} &= \text{softmax}(\mathbf{a}) \end{aligned} \tag{1.10}$$

where the *softmax* function is a convenient way to output a normalized distribution over K classes:

$$\text{softmax}(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{k=1}^{K} e^{z_k}} \tag{1.11}$$

A neural network is usually represented as in Figure 1.3.

Backpropagation algorithm

The gradient of a neural network's loss function with respect to its parameters is computed via the *backpropagation algorithm* (Rumelhart et al., 1988), which efficiently makes use of the chain rule. [i]

The algorithm works as follows:

1. Compute the gradient of the loss with respect to the network's output ($\frac{\partial \mathcal{L}}{\partial \mathbf{y}}$).
2. Compute the gradient of the loss with respect to the last hidden layer as

$$\frac{\partial \mathcal{L}}{\partial \mathbf{h}_M} = \frac{\partial \mathcal{L}}{\partial \mathbf{y}} \frac{\partial \mathbf{y}}{\partial \mathbf{h}_M}. \tag{1.12}$$

3. For hidden layers $M-1$ to 1, compute the gradient of the loss with respect to that layer as

$$\frac{\partial \mathcal{L}}{\partial \mathbf{a}_{j+1}} = \frac{\partial \mathcal{L}}{\partial \mathbf{h}_{j+1}} \frac{\partial \mathbf{h}_{j+1}}{\partial \mathbf{a}_{j+1}}, \quad \text{then} \quad \frac{\partial \mathcal{L}}{\partial \mathbf{h}_j} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}_{j+1}} \frac{\partial \mathbf{a}_{j+1}}{\partial \mathbf{h}_j} \tag{1.13}$$

   by re-using the gradient with respect to the layer above.

i. The backpropagation algorithm is a special case of the more general reverse-mode automatic differentiation algorithm (Griewank and Walther, 2008), which handles more exotic cases like control-flow statements.

4. Gradients with respect to parameters of the layer j are computed as

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial \mathbf{b}_j} &= \frac{\partial \mathcal{L}}{\partial \mathbf{a}_j} \frac{\partial \mathbf{a}_j}{\partial \mathbf{b}_j}, \\ \frac{\partial \mathcal{L}}{\partial W_j} &= \frac{\partial \mathcal{L}}{\partial \mathbf{a}_j} \frac{\partial \mathbf{a}_j}{\partial W_j}. \end{aligned} \tag{1.14}$$

The gradient thus propagates backwards from layer to layer in the network.

1.1.5 Convolutional neural networks

One category of artificial neural networks that is particularly useful for computer vision is the *convolutional neural network* (CNN).

Anatomy of a CNN

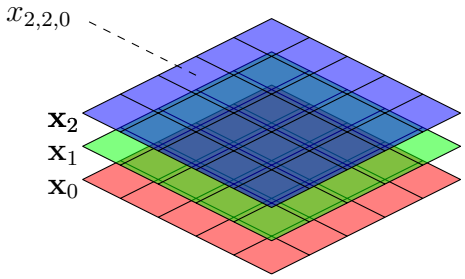
A layer in a CNN is split into groups called *feature maps*. Units within a feature map are laid on a grid, like pixels in an image, and feature maps are stacked one onto another such that units of all feature maps at the ith row and jth column share the same spatial location. We denote the unit of the ith feature map located at row j and column k as $x_{i,j,k}$.

As an example, consider a color image (Figure 1.4a). It is composed of three feature maps: the red channel, the green channel, and the blue channel. Units at the ith row and jth column of those three feature maps form a pixel: each of the three units measures something different about the same spatial location.

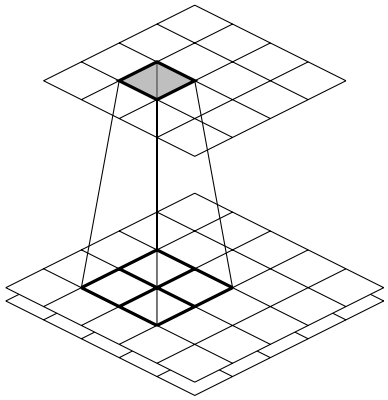
Convolution

To go from one CNN layer to the next, we *convolve* [i] an affine transformation over the layer. This involves sliding an $N \times M \times J \times K$ *kernel* W (where N is the number of input feature maps, M is the number of output feature maps, and J and K are the height and width of the kernel) across the input feature maps and computing the weighted sums defined by the kernel (Figure 1.4b):

i. The operation is actually cross-correlation, but for the purpose of CNNs it makes little difference, as the parameters are learned anyways.



(a) Color image. The red, green and blue channels correspond to the feature maps $\mathbf{x}_0$, $\mathbf{x}_1$ and $\mathbf{x}_2$, respectively. The pixel at the first row and third column has (R,G,B) values $(x_{0,2,0}, x_{1,2,0}, x_{2,2,0})$.



(b) Convolution from two input feature maps to one output feature map. The grey output pixel is computed as a weighted sum of a 2×2 window of units at the same location across all input feature maps. The window slides over the input feature maps to produce the remaining 15 output values, but the weights remain the same.

Figure 1.4 – Convolutional neural networks.

$$h_{m,r,s} = \sum_{i=0}^{N-1} \sum_{a=0}^{J-1} \sum_{b=0}^{K-1} W_{i,m,a,b} x_{i,r+a,s+b} + b_m \tag{1.15}$$

where b_m is the bias for the m^{th} output feature map. The resulting set of feature maps is passed through an elementwise nonlinearity, just as with regular neural networks.

This is motivated by two intuitions:

— Natural images exhibit characteristics that can be described in terms of local features. For instance, a square can be described as a group of straight edges interacting together (forming lines or corners). Because of that, we can drastically reduce the number of free parameters by introducing *sparse connectivity* in the form of local feature detectors.

— We can re-use local feature detectors at different locations in an image: a vertical edge is detected the same no matter where it is located in an image. This allows us to reduce the number of free parameters even further through *weight sharing*.

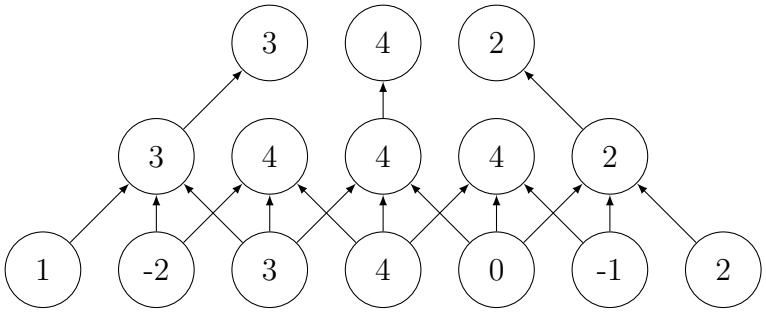


Figure 1.5 – 1D example of pooling and subsampling. Input units are max-pooled in groups of 3, and pooled units are subsampled by a factor of 2.

Pooling and subsampling

In classifying images, we are more interested in deciding whether a particular type of object is present than its exact location. We can therefore benefit from building translational invariance in the network through *pooling*.

Pooling works by summarizing every $n \times m$ group of units in a feature map, either by taking their mean or the maximum value within the group (Figure 1.5).

Since pooled units aggregate information from a group of units, it's oftentimes not necessary to consider all pooled units. Instead, we *subsample* by retaining only every k^{th} unit of the pooled feature maps.

1.2 Probabilistic graphical models

Probabilistic graphical models (PGM) are a way to encode a distribution over a set of random variables as a graph. This representation can potentially be much more compact than the usual probability table. [i]

This is due to the fact that PGMs encode dependencies between random variables as edges or arrows between nodes. When there are a lot of conditional independences, the probability table contains a lot of redundant information which PGMs help eliminate (Figure 1.6).

i. This section is adapted in part from a blog post on variational autoencoders (Dumoulin, 2014).

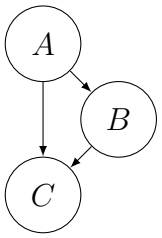
(a, b, c)	$P(A = a, B = b, C = c)$
(0, 0, 0)	3/32
(0, 0, 1)	1/32
(0, 1, 0)	9/32
(0, 1, 1)	3/32
(1, 0, 0)	3/32
(1, 0, 1)	1/32
(1, 1, 0)	9/32

(a) The probability table contains a lot of reduntant information.

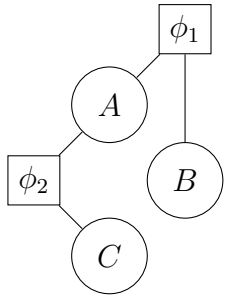
X	$P(X = 0)$
A	1/2
B	1/4
C	3/4

(b) Knowing that A, B and C are independent, the distribution can be represented much more compactly.

Figure 1.6 – A distribution over independent variables A, B and C.



(a) Bayesian network for $P(A, B, C) = P(A)P(B \mid A)P(C \mid A, B)$.



(b) Markov network for $P(A, B, C) = \frac{1}{Z}\phi_1(A, B)\phi_2(A, C)$.

Figure 1.7 – Graphical representation of Bayesian and Markov networks.

There are two types of PGMs: *directed PGMs* and *undirected PGMs*. Both rely on a graph in which random variables are represented as nodes and dependencies between random variables are represented as edges (or arrows).

1.2.1 Directed probabilistic graphical models

Directed PGMs, also known as *Bayesian networks*, rely on directed acyclic graphs. Distributions they encode are of the form

$$P(X_1, X_2, \ldots, X_N) = \prod_{i=1}^{N} P(X_i \mid \mathrm{Pa}(X_i)) \tag{1.16}$$

where $\text{Pa}(X_i)$ is a subset of $\{X_1, \ldots, X_n\} \setminus \{X_i\}$ on which X_i depends directly.

To go from a distribution to its graphical representation, each random variable is mapped to a node in the graph, and an arrow between each node in $\text{Pa}(X_i)$ and X_i is added for all X_i (Figure 1.7a).

1.2.2 Undirected probabilistic graphical models

Undirected PGMs, also known as *Markov networks*, rely on undirected graphs. Distributions they encode are of the form

$$\begin{aligned} P(X_1, X_2, \ldots, X_N) &= \frac{1}{Z}\tilde{P}(X_1, X_2, \ldots, X_N), \\ \tilde{P}(X_1, X_2, \ldots, X_N) &= \prod_{i=1}^{m} \phi_i(\mathbf{D}_i) \end{aligned} \tag{1.17}$$

where

$$Z = \sum_{X_1, \ldots, X_N} \tilde{P}(X_1, X_2, \ldots, X_N) \tag{1.18}$$

is the *partition function* making sure the distribution is normalized, $\phi_i(\mathbf{D}_i)$ is a non-negative function called *factor*, and $\mathbf{D}_i$ is a subset of $\{X_1, \ldots, X_N\}$.

To go from a distribution to its graphical representation, each factor is mapped to a "factor" node in the graph, each random variables is mapped to a "variable" node in the graph, and an edge is drawn between a variable node and a factor node if the corresponding variable appears in the corresponding factor (Figure 1.7b).

1.3 Generative adversarial networks

Generative adversarial networks (Goodfellow et al., 2014) are a family of generative models which rely on two networks, the generator network and the discriminator network, to implicitly learn the data-generating distribution. Informally, the generator is tasked with mapping a source of random noise to samples which the discriminator believes were sampled from the data-generating distribution, while

the discriminator is tasked with correctly classifying the samples it receives as being either from the generator distribution (label 0) or the data-generating distribution (label 1).

More formally, let $q(\mathbf{x})$ be the data-generating distribution, and let $p(\mathbf{x})$ be the generator distribution implicitly defined by

$$\mathbf{x} = G(\mathbf{z}), \quad \mathbf{z} \sim p(\mathbf{z}), \tag{1.19}$$

where $\mathbf{z}$ is the source of random noise, $p(\mathbf{z})$ is its distribution, and $G(\mathbf{z})$ is an arbitrary function (usually a neural network) called the *generator network* mapping $\mathbf{z}$ to $\mathbf{x}$.

Moreover, let $C = 0$ be the event $\mathbf{x} \sim p(\mathbf{x})$ and $C = 1$ be the event $\mathbf{x} \sim q(\mathbf{x})$, and let $D(\mathbf{x})$ be the *discriminator network*, whose output is interpreted as

$$D(\mathbf{x}) = P(C = 1). \tag{1.20}$$

Finally, let

$$V(D, G) = \mathbb{E}_{q(\mathbf{x})}[\log D(x)] + \mathbb{E}_{p(\mathbf{z})}[\log(1 - D(G(\mathbf{z})))] \tag{1.21}$$

be the *value function* minimized and maximized by the generator and discriminator networks, respectively.

Then the optimization problem

$$\min_G \max_D V(D, G) \tag{1.22}$$

has the following properties:

— For a fixed G, maximizing $V(D, G)$ with respect to D produces a discriminator whose output can be interpreted as the Jensen-Shannon divergence (Lin, 1991) between $q(\boldsymbol{x})$ and $p(\boldsymbol{x})$.

— The global minimum of $\max_D V(D, G)$ with respect to G is attained if and only if $p(\mathbf{x}) = q(\mathbf{x})$ for all $\mathbf{x}$, at which point $V(D, G) = -\log 4$.

In other words, in the non-nonparametric limit and assuming D and G are optimized in function space, alternating between maximizing $V(D, G)$ with respect to D and doing one gradient step to minimize $V(D, G)$ with respect to G will converge to the generator distribution being equal to the data-generating distribution.

In practice, minimizing $V(D, G)$ with respect to G is replaced with maximizing

$$\tilde{V}(D, G) = \mathbb{E}_{p(\mathbf{z})}[\log D(G(\mathbf{z}))] \tag{1.23}$$

for optimization reasons. Furthermore, D and G are trained in parameter space using simultaneous updates, which has been shown by Nagarajan and Kolter (2017) (under suitable conditions) to be an optimization procedure for which equilibrium points are locally asymtotically stable, meaning that for some region around an equilibrium point the optimization procedure converges at an exponential rate to that equilibrium point.

1.4 Energy-based models

1.4.1 Definition

Energy-based models form a commonly used group of undirected probabilistic graphical models. Their distribution has the form

$$p(\mathbf{s}) = \frac{\exp\left(-E(\mathbf{s})\right)}{Z} \tag{1.24}$$

where $E(\mathbf{s})$ is the *energy function* and Z is the usual undirected PGM partition function, i.e.

$$Z = \sum_{\tilde{\mathbf{s}}} \exp\left(-E(\tilde{\mathbf{s}})\right).^{\text{i}} \tag{1.25}$$

Configurations with lower energy are more probable, and configurations with higher energy are less likely. We are interested in finding a parametrization for the energy function which minimizes the energy of training examples compared to other configurations.

i. We suppose that $\mathbf{s}$ is discrete for simplicity, but the identities that follow also hold for the continuous case.

1.4.2 Training

Training an energy-based model can be done by maximizing the log-likelihood of the training set.

When $\mathbf{s}$ is fully observed, the gradient of the log-likelihood with respect to the set θ of model parameters is

$$\frac{\partial}{\partial\theta}\log p(\mathbf{s}) = -\frac{\partial}{\partial\theta}E(\mathbf{s}) + \mathbb{E}_{p(\tilde{\mathbf{s}})}\left[\frac{\partial}{\partial\theta}E(\tilde{\mathbf{s}})\right]. \tag{1.26}$$

The gradient decomposes into two terms:

1. The positive phase term $-\frac{\partial}{\partial\theta}E(\mathbf{s})$ wants to decrease the energy of an example $\mathbf{s}$, and thus increase its probability.
2. The negative phase term $\mathbb{E}_{p(\tilde{\mathbf{s}})}\left[\frac{\partial}{\partial\theta}E(\tilde{\mathbf{s}})\right]$ wants to increase the energy of all other configurations, and thus decreases their probability.

In the fully observed case, the positive phase is tractable, but the negative phase is intractable for all but the smallest models: the number of configurations over which to sum to compute the partition function increases exponentially with the dimensionality of $\mathbf{s}$. The negative phase is usually approximated by Monte Carlo:

$$\mathbb{E}_{p(\tilde{\mathbf{s}})}\left[\frac{\partial}{\partial\theta}E(\tilde{\mathbf{s}})\right] \approx \frac{1}{N}\sum_{i=1}^{N}\frac{\partial}{\partial\theta}E(\tilde{\mathbf{s}}_i), \quad \tilde{\mathbf{s}}_i \sim p(\tilde{\mathbf{s}}) \tag{1.27}$$

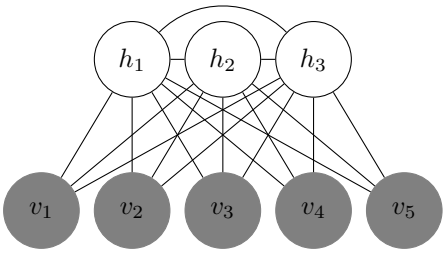
When $\mathbf{s}$ is partially observed, the probability distribution can be written as

$$p(\mathbf{v},\mathbf{h}) = \frac{\exp\left(-E(\mathbf{v},\mathbf{h})\right)}{Z}, \tag{1.28}$$

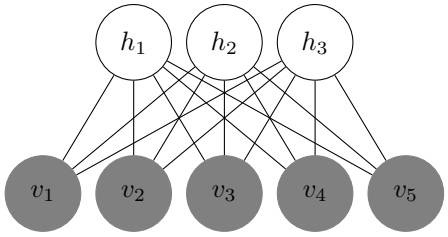
where $\mathbf{s}$ has been partitioned into a set $\mathbf{v}$ of visible (or observed) variables and a set $\mathbf{h}$ of hidden (or latent) variables.

In the partially observed case, the log-likelihood gradient is

$$\begin{aligned}\frac{\partial}{\partial\theta}\log p(\mathbf{v}) &= \frac{\partial}{\partial\theta}\log\left(\sum_{\tilde{\mathbf{h}}}p(\mathbf{v},\tilde{\mathbf{h}})\right)\\ &= -\mathbb{E}_{p(\tilde{\mathbf{h}}|\mathbf{v})}\left[\frac{\partial}{\partial\theta}E(\mathbf{v},\tilde{\mathbf{h}})\right] + \mathbb{E}_{p(\tilde{\mathbf{v}},\tilde{\mathbf{h}})}\left[\frac{\partial}{\partial\theta}E(\tilde{\mathbf{v}},\tilde{\mathbf{h}})\right],\end{aligned} \tag{1.29}$$



(a) Boltzmann machine. The edges between h_1, h_2 and h_3 make the positive phase of the gradient intractable.



(b) Restricted Boltzmann machine. Nodes form a bipartite graph, with $\mathbf{v}$ and $\mathbf{h}$ being the two partition sets.

Figure 1.8 – Boltzmann machines over a set $\mathbf{v} = (v_1, v_2, v_3, v_4, v_5)$ of observed variables and a set $\mathbf{h} = (h_1, h_2, h_3)$ of latent variables.

and the positive phase also becomes intractable except for small models and particular architectures like the restricted Boltzmann Machine (RBM).

1.4.3 Sampling

Samples can be drawn from some energy-based models using *Gibbs sampling*.

The state $\mathbf{s}$ is first initialized at a random value of $\mathbf{s}^{(0)}$. We then successively sample $\mathbf{s}^1, \ldots, \mathbf{s}^T$ by doing T *Gibbs steps*.

A Gibbs step consists of successively sampling variables in $\mathbf{s}$ given all other variables, i.e.

$$s_i^{(t+1)} \sim p(s_i \mid s_0^{(t+1)}, \ldots s_{i-1}^{(t+1)}, s_{i+1}^{(t)}, \ldots, s_{|\mathbf{s}|}^{(t)}), \quad i = 1, \ldots, |\mathbf{s}|. \tag{1.30}$$

At the end of the T Gibbs steps, a sample is obtained. T has to be large enough to allow for the sampler to *burn in*: it takes some time before the sampler reaches equilibrium, and initial samples won't represent the desired distribution well. The time required to reach equilibrium is dependent on the initial state but can be formally defined in the worst-case scenario as the *mixing time*. When we say that our sampler "mixes well", we loosely mean that it reaches equilibrium quickly.

Once the first sample is obtained, subsequent samples can be drawn by doing N Gibbs steps starting from the previous sample, with the appropriate value of N depending on how well the model mixes. In practice it is common to simply run multiple chains in parallel to obtain multiple samples.

The procedure can be accelerated through *block Gibbs sampling*. Instead of

sampling one variable at a time, $\mathbf{s}$ is partitioned into sets of variables that are conditionally independent given all other variables, and all variables in a partition set are sampled at once.

1.4.4 Boltzmann machines

When the energy function is quadratic, the model is called a *Boltzmann machine*:

$$E(\mathbf{s}) = -\mathbf{s}^T W \mathbf{s} - \mathbf{s}^T \mathbf{b} \tag{1.31}$$

Boltzmann machines in general are not guaranteed to have a tractable positive phase in the gradient, because nothing forces the latent variables to be conditionally independent given the observed variables, i.e. there may be edges between latent nodes in the graph (Figure 1.8a).

1.4.5 Restricted Boltzmann machines

When we add to a Boltzmann machine the constraint that there can be no edges between visible pairs or between latent pairs of nodes, the model becomes a *Restricted Boltzmann machine* (RBM) (Smolensky, 1986). Its energy function has the form

$$E(\mathbf{v}, \mathbf{h}) = -\mathbf{b}^T \mathbf{v} - \mathbf{c}^T \mathbf{h} - \mathbf{h}^T W \mathbf{v} \tag{1.32}$$

The graphical representation of an RBM is a bipartite graph with $\mathbf{v}$ and $\mathbf{h}$ being the two partition sets (Figure 1.8b).

For $\mathbf{v} \in \{0,1\}^{|\mathbf{v}|}$ and $\mathbf{h} \in \{0,1\}^{|\mathbf{h}|}$, the positive phase of an RBM's gradient is

$$-\mathbb{E}_{p(\tilde{\mathbf{h}}|\mathbf{v})}\left[\frac{\partial}{\partial\theta} E(\mathbf{v}, \tilde{\mathbf{h}})\right] = -\frac{\partial}{\partial\theta}\mathcal{F}(\mathbf{v}) \tag{1.33}$$

where

$$\mathcal{F}(\mathbf{v}) = -\mathbf{b}^T \mathbf{v} - \sum_{j=1}^{|\mathbf{h}|} \log(1 + \exp(c_j + W_j \mathbf{v})) \tag{1.34}$$

is the *free energy* of $\mathbf{v}$.

Sampling from an RBM is done by block Gibbs sampling, alternating between sampling from $\mathbf{v}$ given $\mathbf{h}$ and vice versa. Conditional distributions are analytically computed as

$$\begin{aligned} p(v_i = 1 \mid \mathbf{h}) &= \sigma(\mathbf{h}^T W_{:,i} + \mathbf{c}), \\ p(h_j = 1 \mid \mathbf{v}) &= \sigma(W_{j,:}\mathbf{v} + \mathbf{b}) \end{aligned} \tag{1.35}$$

1.4.6 Training an RBM

Even though its positive phase is tractable, the RBM has an intractable negative phase (Long and Servedio, 2010) that has to be approximated via sampling-based methods, such as *contrastive divergence* (CD) (Hinton, 2002; Hinton et al., 2006) or *persistent contrastive divergence* (PCD) (Younes, 1999; Tieleman, 2008).

In CD, samples are drawn by initializing visible configurations with training examples and doing one Gibbs step. The CD-k variant of this method does k Gibbs steps instead of one. In doing so, we increase the energy of configurations near training examples. Coupled with the positive phase, it has the effect of creating energy 'wells' around training examples. Even though CD performs well in practice, the parameter updates it yields do not correspond to the gradient of any function (Sutskever and Tieleman, 2010).

In PCD, the state of a set of *fantasy particles* is made to persist across parameter updates and samples are drawn by doing one Gibbs step. As with CD-k, PCD-k is a variant of PCD which does k Gibbs steps instead of one. The idea behind PCD is that if the fantasy particles are near equilibrium and a small parameter update is made, it will not take too many Gibbs steps to reach equilibrium again.

2 Prologue to First Article

2.1 Article Details

On the Challenges of Physical Implementations of RBMs.

Vincent Dumoulin, Ian J. Goodfellow, Aaron Courville, and Yoshua Bengio. *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 1199-1205.

Personal Contribution.

I am first contributor to this work with regards to experiment design, analysis and writing, and I was in charge of carrying out the experiments.

2.2 Context

We consider the use of physical hardware to train RBMs. Although RBMs are powerful machine learning models, learning and some kinds of inference in the model require sampling-based approximations, which, in classical digital computers, are implemented using expensive MCMC. Physical computation offers the opportunity to reduce the cost of sampling by building physical systems whose natural dynamics correspond to drawing samples from the desired RBM distribution. Such a system avoids the burn-in and mixing cost of a Markov chain. However, hardware implementations of this variety usually entail limitations such as low-precision and limited range of the parameters and restrictions on the size and topology of the RBM.

2.3 Contributions

We conduct software simulations to determine how harmful each of these restrictions is. Our simulations are based on the D-Wave Two computer, but the issues we investigate arise in most forms of physical computation.

Our findings suggest that designers of new physical computing hardware and algorithms for physical computers should focus their efforts on overcoming the limitations imposed by the topology restrictions of currently existing physical computers.

2.4 Recent Developments

Physical implementation of machine learning models, and more specifically Boltzmann machines, remains a niche research area to this day — notable examples include NIPS 2015's Quantum Machine Learning Workshop.

Recent research efforts in the deep learning community have shifted from energy-based models to explicit or implicit directed probabilistic graphical models (PGMs), which can be sampled from easily through ancestral sampling. The training of explicit directed PGMs like variational autoencoders (Kingma and Welling, 2014; Rezende et al., 2014) and various flavours of autoregressive models (Larochelle and Murray, 2011; Germain et al., 2015; van den Oord et al., 2016c,b,a) has been made practical by recent advances in stochastic inference (for latent-variable models) and the introduction of highly parallelizable autoregressive structures (for autoregressive models). Implicit directed PGMs have been spearheaded by generative adversarial networks (GANs) (Goodfellow et al., 2014), which take a radically different approach to obtaining gradients on the directed PGM model, and which will be expanded upon in more detail in chapter 5. Of note for this article is the recently-proposed GibbsNet model (Lamb et al., 2017), which can be thought of as an energy-based model defined *implicitly* through the conditionals $q(\mathbf{z} \mid \mathbf{x})$ and $p(\mathbf{x} \mid \mathbf{z})$ and which is trained using an adversarial framework.

3 On the Challenges of Physical Implementations of RBMs

3.1 Introduction

A restricted Boltzmann machine, as discussed in subsection 1.4.5, is a generative model that has found widespread application (Hinton et al., 2006; Bengio, 2012; Coates and Ng, 2011). At the time of this work, RBMs remain part of the state of the art system for classifying permutation invariant MNIST (Hinton et al., 2012). RBMs and other Boltzmann machines are the dominant means of using deep learning to solve tasks that involve unsupervised learning and probabilistic modeling, such as filling in missing values or classification with missing inputs (Goodfellow et al., 2013a). Unfortunately, the log likelihood of the RBM is intractable (Long and Servedio, 2010), and for other Boltzmann machines most other interesting quantities are intractable as well. In this work, we explore the use of quantum hardware to overcome these difficulties. This approach could possibly unlock the untapped potential of non-restricted Boltzmann machines.

The model may be trained using sampling-based approximations to the gradient of the log likelihood (Younes, 1999; Tieleman, 2008). However, drawing a fair sample from the model is also intractable (Long and Servedio, 2010).

Drawing samples from an RBM on a classical digital computer is an active area of research (Salakhutdinov, 2010; Desjardins et al., 2010; Cho et al., 2010). Existing approaches are based on Markov chain Monte Carlo (MCMC) procedures. The cost of drawing a fair sample using an MCMC method may be high if the number of steps required to get a good sample is high. This occurs in practice because some RBMs represent distributions with modes that are separated by regions of extremely low probability, which the Markov chain crosses only rarely. This is particularly problematic because it interacts with the learning procedure in a vicious circle: as training progresses, parameters (weights and biases) gradually become larger, corresponding to sharper probabilities (higher near training examples, and

smaller elsewhere), i.e., corresponding to sharper modes separated by zones of lower probability. Since training procedures based on approximating the log-likelihood gradient require sampling from the model (usually by MCMC), as training progresses sampling becomes more difficult (mixing more slowly between modes, i.e., more samples would be required to achieve the same level of variance in the MCMC estimator of the gradient), making the gradient less reliable and thus slowing down training.

One possible solution is to construct a physical system whose natural behavior is to take on states with the desired probability. One may then obtain the desired samples by observing the behavior of the system, rather than explicitly performing computations to simulate the dynamics of such a system. We refer to this approach as "physical computation". It is similar in spirit to "analog computation" but we find that term inappropriate in this case, since the sampled states remain digital. Note that this is different from the idea of building an RBM "in hardware"–we are not merely advocating designing an FPGA that specializes in performing the kinds of digital computations used for simulating an RBM.

Physical computation is a strategy being actively pursued by D-Wave Systems Inc. [i] and DARPA's UPSIDE program [ii]. In particular, the D-Wave Two system can be viewed as a physical implementation of an RBM. Most approaches to physical computation share the property that they greatly simplify the complexity of a task that is difficult for digital computers, but also introduce many limitations that digital computers do not share. For instance, any physical implementation of an RBM will likely face the issues of *noisy parameters*, *limited parameter range* and *restricted architecture*. This work aims at getting a better understanding of the effect of these three constraints on the training and performance of the physical RBM and ultimately, of the feasibility of the physical approach. In particular, we would like to address the following questions:

— Which constraint has the worst effect on performance?

— Under which circumstances can a physical implementation of the RBM be reasonably trained?

— Are there ways to mitigate the degrading effects of constraints imposed by

i. http://www.dwavesys.com/en/products-services.html

ii. http://www.darpa.mil/Our_Work/MTO/Programs/Unconventional_Processing_of_Signals_for_Intelligent_Data_Exploitation_(UPSIDE).aspx

physical computation?

Currently, the only practical physical RBM available is the D-Wave Two system (but see (Dupret et al., 1996) for earlier work on physical computation also associated with Ising models). It suffers from all three of the limitations we wish to study. In order to study each limitation in isolation, we performed a suite of feasibility studies using a *simulated* physical computer, that we implemented in software on a GPU. Using a simulation allows us to observe what happens when a physical computer has noisy parameters, but not limited parameter range or architecture restrictions, etc. Because these experiments are performed in simulation, we do not capture the *benefit* of physical computation: faster, less correlated samples. Instead, we aim to characterize the potential *detriments* of physical computation. In particular, by studying each constraint in isolation, we are able to infer their relative effect on performance and thereby offer guidance for how both hardware and algorithm designers can best focus their efforts on those properties of physical computation that impose the greatest barriers to its practical use.

3.2 RBM training challenges

As discussed in subsection 1.4.5, the negative phase of an RBM's gradient is obtained via a sampling-based approximation. Although conditional sampling in an RBM is trivial, sampling from $p(\mathbf{v}, \mathbf{h})$ or from $p(\mathbf{v})$ cannot be done in a single step and requires the use of Monte Carlo Markov chains, which in general becomes computationally expensive if the parameters $\mathbf{W}$, $\mathbf{b}$, and $\mathbf{c}$ are configured in a way that makes the Markov chain mix slowly.

The persistent contrastive divergence (PCD) algorithm (also known as stochastic maximum likelihood) (Younes, 1999; Tieleman, 2008) uses a persistent Gibbs (MCMC) sampling scheme that sequentially samples from the conditionals $p(\mathbf{h} \mid \mathbf{v})$ and $p(\mathbf{v} \mid \mathbf{h})$ to recover samples from the joint distribution. These samples are then used in a Monte Carlo approximation of the negative phase contribution of the log likelihood gradient.

While PCD has established itself as probably the most popular method of maximizing log likelihood in RBMs, it suffers from one important weakness. In many situations, as learning progresses and the model parameters begin to increase in

magnitude, the Gibbs sampler at the heart of the negative phase contribution of the gradient can suffer from poor mixing properties. Generally, it occurs when the hidden and visible activations become highly correlated. Poor mixing in the Gibbs sampling induced Markov chain leads to poor sample diversity which in turn leads to poor estimates of the negative phase statistics which ultimately lead to a poor approximation of the likelihood gradient. This problem can be somewhat mitigated by increasing sample diversity through the use of PCD-k (using k Gibbs sampling steps between gradient updates). Other ways to mitigate the negative phase mixing issue include the use of auxilliary parameters (Tieleman and Hinton, 2009) and tempering methods (Salakhutdinov, 2009; Desjardins et al., 2010; Cho et al., 2010).

The promise of a physical implementation of the RBM is that we entirely sidestep the difficult mixing problem that occurs in the negative phase of training by aquiring fair, uncorrelated samples directly from a physical implementation of the RBM. In the next section we review the D-Wave machine, to our knowledge the only existing physical implementation of an RBM-like model.

3.3 The D-Wave system

The D-Wave Two system implements an *Ising model* (Ising, 1925). Specifically, it has a signed state vector $s \in \{-1, 1\}^{512}$ and a quadratic energy function

$$E(s) = s^T J s + g^T s$$

where J is analogous to the weights of a Boltzmann machine and g is analogous to its biases. The set of Ising model distributions with $\{-1, 1\}$ states is isomorphic to the set of Boltzmann machine distributions with $\{0, 1\}$ states. The conversion between the parameters of the two model families is a linear mapping. An RBM with $\{0, 1\}$ states h and v encoded with weights W and biases b and c can be converted to use $\{-1, 1\}$ states via the mapping:

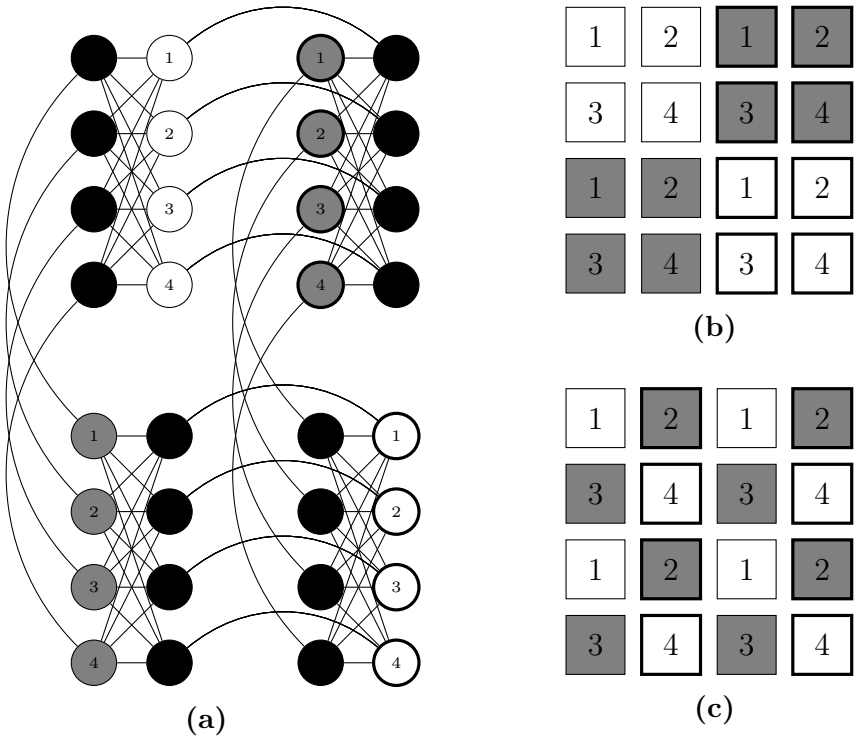


Figure 3.1 – Two different ways of mapping pixels of an image to visible units of an RBM with chimera connectivity (3.1a). *Pixel blocks* (3.1b) involves mapping adjacent 2×2 blocks of pixels to adjacent, fully-connected groups of units while respecting the relative positions of blocks of pixels. *Extended pixel blocks* (3.1c) makes the pixel blocks overlap, to capture more long-range relationships.

$$\begin{aligned} W' &= \frac{W}{4} \\ b'_i &= \frac{1}{2} b_i + \frac{1}{4} \sum_j W_{ij} \\ c'_i &= \frac{1}{2} c_i + \frac{1}{4} \sum_j W_{ji} \end{aligned} \tag{3.1}$$

One can draw samples from a Boltzmann machine using the D-Wave Two system just by performing this linear conversion of the parameters prior to requesting the sample. The resulting $\{-1, 1\}$ sample may be converted to a $\{0, 1\}$ sample simply by replacing all instances of -1 with 0. The choice of parameterization affects the learning dynamics of stochastic gradient descent, and the Boltzmann

parameterization is usually better, so it is generally best to regard the model as a Boltzmann machine even if the interface to the sampling hardware uses the Ising parameterization.

The actual probability distribution sampled by the D-Wave Two system deviates slightly from $p(s) \propto \exp(-E(s))$. Moreover, it is difficult to control the value of J or g precisely. Both effects can be approximated by adding Gaussian noise to J and g. To simulate the D-Wave Two system with reasonable accuracy, the noise should be added to J once each time the value of J is changed to a new unique value, but the noise on g should be resampled every time a new sample is drawn[i]. This is the approach we take in our GPU-based simulator of the D-Wave Two system. (One complication we do not attempt to model is that if the same value of J is requested twice, the error on J should be the same both times–it is not truly noise, but rather a deterministic error that has a Gaussian distribution when compared over multiple points in J space.) Other approaches to physical computation, such as those explored by DARPA's UPSIDE program, face similar issues with noise.

The D-Wave Two system also imposes restrictions on the magnitude of each individual element of J and g. This is common to most approaches to physical computation.

Finally, many elements of J are constrained to be zero. This is because the various elements of the state vector are physically laid out in a 2-D grid, and only nearby elements can interact with each other. Specifically, the connectivity of the graphical model is constrained to be a chimera graph as illustrated in Figure 3.1. We observe that this chimera graph can be partitioned to form a bipartite graph. Under such a partition, the D-Wave Two system comes very close to being an RBM. The only difference between this model and an RBM is that the noise on the biases causes the biases to be random variables rather than parameters of the model.

Denil and De Freitas have also explored the use of D-Wave hardware for training RBMs. Like our work, their work is primarily a feasibility study based on software simulations. Their approach differs from ours in three respects: 1) We partition the D-Wave Two system into visible and hidden states using a partitioning that makes the chimera graph bipartite, so the hardware implements an RBM. Denil

i. Andrew Berkley, D-Wave Principal Scientist, personal communication

and De Freitas used a different partitioning that allowed visible-visible and hidden-hidden interactions. 2) We train using sampling-based approximations to the log likelihood gradient, while they train using empirical derivatives of an autoencoder-like cost function. 3) Our focus is on understanding how detrimental each of the limitations of the D-Wave hardware is in isolation, while Denil and De Freitas focus on devising an algorithm that works reasonably well with all limitations in place simultaneously.

3.4 Methodological notes

All models were trained using PCD-15. We used standard train / test split for the MNIST (LeCun et al., 1998), Connect-4 and OCR Letters (Larochelle et al., 2010) datasets. For all experiments involving training on the simulated physical computer, we used the simulator to draw samples for the negative phase of PCD, but used exact mean field for the positive phase. Training examples were binarized every time they were presented by sampling from a Bernoulli distribution, such that the grayscale value in [0, 1] in the original image gives the probability of that pixel being a 1 in the binary image. Unless explicitly stated, all models were trained using the same hyperparameters.

Negative log-likelihood (NLL) of all models is approximated using annealed importance sampling (AIS) (Salakhutdinov and Murray, 2008). When noise is added to parameters, the expected AIS is computed by Monte Carlo, with test examples binarized by following the same method as with training examples.

Although the constraints we apply are dictated by the D-Wave Two system, we are simulating a low-precision RBM, which means that constraints are enforced on parameters directly, without converting them to the Ising parametrization first.

All images of samples are displaying the *expected value* of the visible units given binary samples of the hidden units.

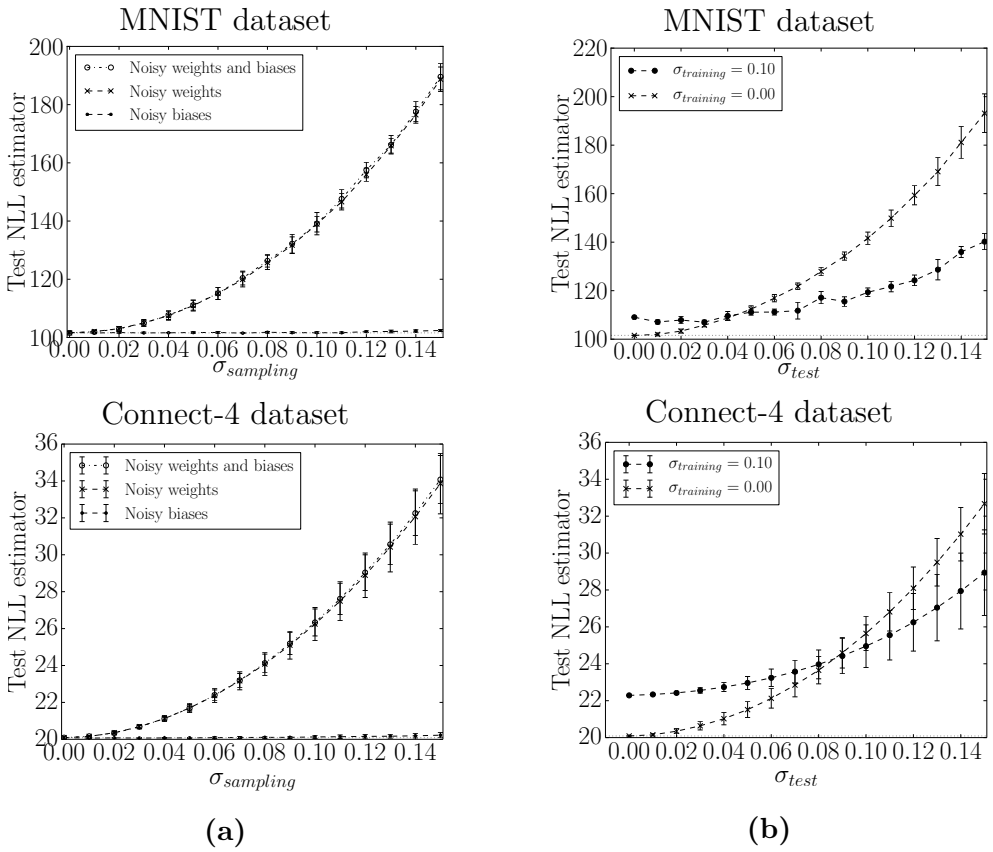
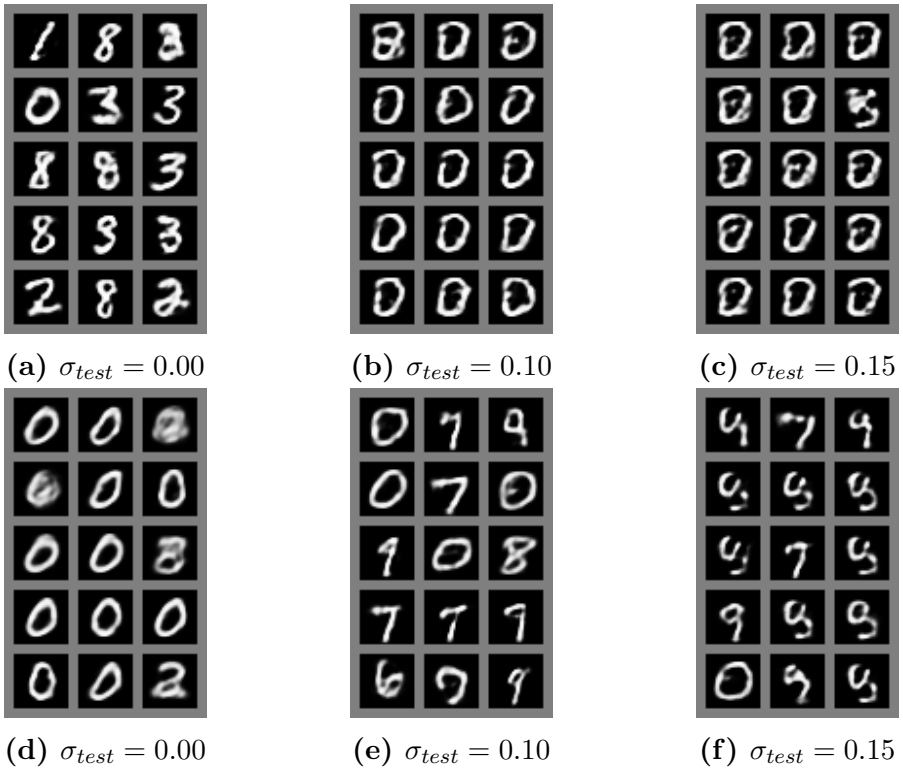


Figure 3.2 – (3.2a) Test negative log likelihood (RBM) estimator of a regularly-trained RBM when Gaussian noise is applied to parameters during sampling. For each noise level, RBM was computed for 5 different seeds. Noise on biases has practically no effect on performance compared to noise on weights. (3.2b) Test RBM estimator of two RBMs trained with different weights and bias noise distributions when Gaussian noise is applied to parameters during sampling. For each noise level, NLL was computed for 5 different seeds. For both experiments, qualitatively similar results were obtained for the OCR Letters dataset but not displayed due to space constraints.

3.5 Simulating noisy parameters

Consider the case where we have a trained RBM (trained by any succesful means; in these experiments we obtained ours by traditional training on a digital computer), and we would like to draw samples from it using physical computation. In this case, we know that the model parameters represent the desired distribution well. However, when loaded into the physical computer, the parameters may not be preserved exactly. We simulate this by adding Gaussian noise to the parameters.

See Figure 3.2a for a summary of the experimental results in this case. We find that noise on biases has a negligible effect on NLL compared to noise on weights.



(a) $\sigma_{test} = 0.00$ **(b)** $\sigma_{test} = 0.10$ **(c)** $\sigma_{test} = 0.15$

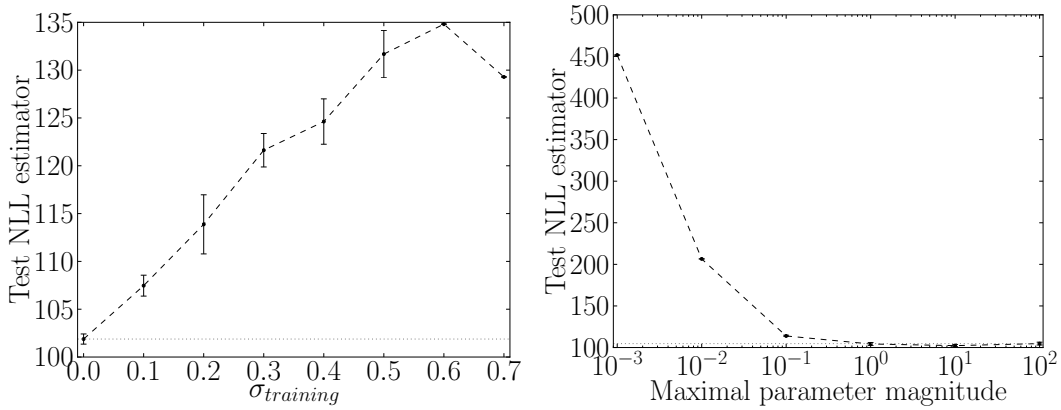
(d) $\sigma_{test} = 0.00$ **(e)** $\sigma_{test} = 0.10$ **(f)** $\sigma_{test} = 0.15$

Figure 3.3 – Random samples after 100,000 Gibbs steps for an RBM trained without noise (top row) and for an RBM trained with Gaussian noise of standard deviation $\sigma = 0.1$ applied to weights and biases (bottom row), for different levels of parameter noise.

This could be explained by the fact there are simply more weight parameters than bias parameters contributing to the energy function. In that case, variance of the energy function would be dominated by variance on weights. From these tests, we can observe two things:

1. Adding noise to the model parameters quickly degrades its performance.
2. Noise on the biases is less harmful than noise on the weights.

Of course, these parameters were trained to work well in the absence of noise. It is possible to learn different parameters, that are chosen to diminish the effect of noise. In order to do this, we trained an RBM using the simulated physical computer to draw the negative phase samples during training. The negative phase repels the model parameters from regions that produced poor samples. Using noise on the parameters while generating the negative phase samples increases the range of the repulsion – not only must the parameters not generate bad samples, noisy



(a) **(b)**

Figure 3.4 – (3.4a) Test NLL estimator computed by sampling with no added parameter noise from RBMs trained with various parameter noise levels. For each noise level, 5 models were trained using the same hyperparameters but different seeds. (3.4b) Test NLL estimator computed by sampling from RBM trained with various magnitude constraints. For each magnitude level, 5 models were trained using the same hyperparameters but different seeds.

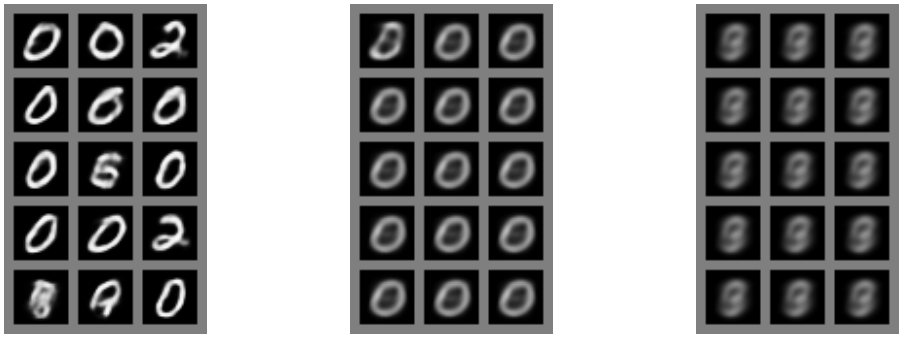
versions of the parameters must not do so either.

We compared how RBM performance evolves as we increase parameter noise during sampling with that of the RBM trained without noisy parameters (Figure 3.2b). We used the same noise distribution for both weights and biases.

We find that training with noisy parameters helps reducing the degrading effect of sampling with noisy parameters. For instance, by training with $\sigma = 0.10$ on parameters and sampling with the same σ, we were able to reduce NLL estimator increase by 21.9% in average when compared to training without noise. Furthermore, the benefits of training with noisy parameters before sampling with noisy parameters extends to noise levels greater than used during training.

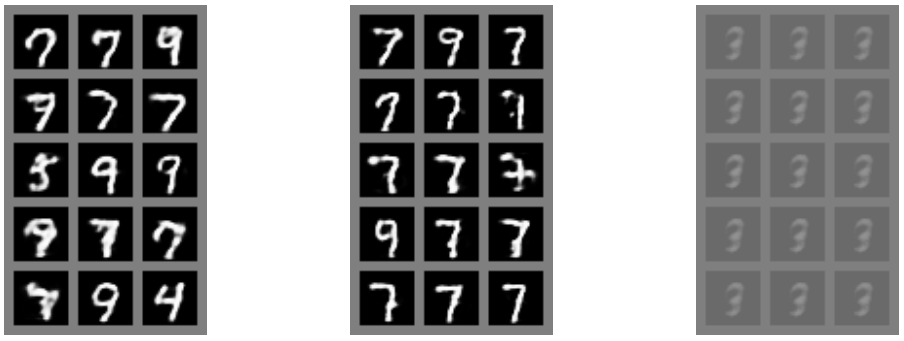
The effect of training with noisy parameters is also qualitatively visible when looking at samples from the model (Figure 3.3). We observe that adding noise to parameters during sampling increases visual noise in samples, and also makes samples collapse to major modes. By training with noisy parameters, we are able to soften these effects, even when sampling with parameter noise greater than training noise.

As for how much parameter noise an RBM can support during training, we trained RBMs using various noise levels on weights and biases and computed their test NLL estimator when sampling with no added noise (Figure 3.4a). A noise



(a) $\sigma_{train} = 0.10$ **(b)** $\sigma_{train} = 0.30$ **(c)** $\sigma_{train} = 0.70$

Figure 3.5 – Random samples after 100,000 Gibbs steps for three different parameter noise levels. Sampling was done without adding noise to parameters.



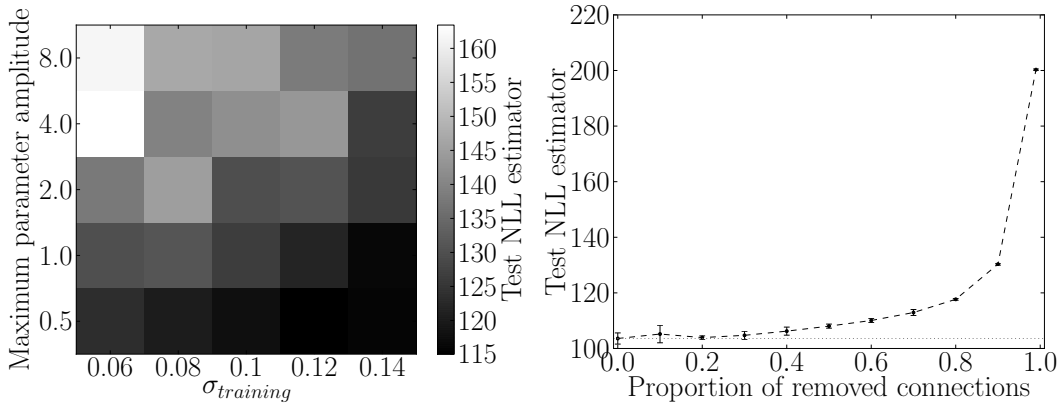
(a) Max. mag. = 100.0 **(b)** Max mag. = 1.0 **(c)** Max mag. = 0.001

Figure 3.6 – Random samples after 100,000 Gibbs steps for three different parameter magnitude constraints.

level of $\sigma = 0.1$ is the biggest noise we could add before the RBM's performance noticeably started to degrade. Figure 3.5 offers a qualitative overview of this effect. This means noisy parameters negatively affects learning for all but the smallest noise values.

3.6 Simulating limited parameter range

We now turn our attention to the parameter range constraint. We trained RBMs by forcing their parameter magnitude to stay below a certain threshold value and observed the effect of that value on test NLL (Figure 3.4b). Whenever



(a) **(b)**

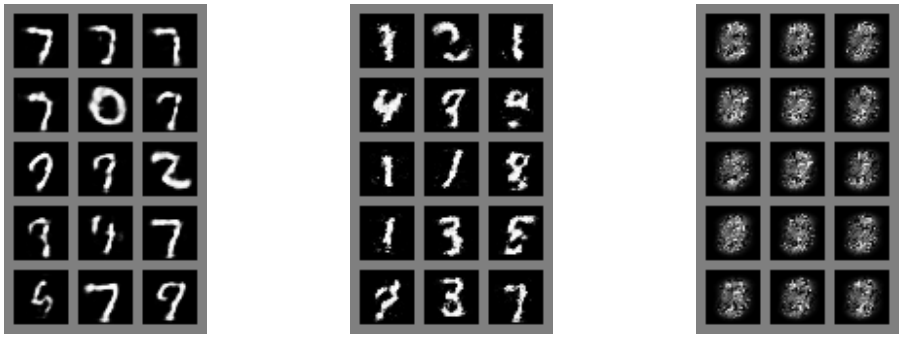
Figure 3.7 – (3.7a) Test NLL estimator for combinations of noise and magnitude constraints. In all cases, the model was evaluated using the same σ as it was trained with. (3.7b) Test NLL estimator computed by sampling from RBMs trained with varying amounts of removed connections (selected at random) in order to simulate a constrained architecture. For each proportion of removed connections, 5 models were trained using the same hyperparameters but different seeds.

parameter updates would bring a parameter outside of that range, it was clipped to the threshold value.

We find that a magnitude constraint higher than or equal to 1.0 has little to no effect on performance, but that forcing parameters' magnitude to be smaller than that quickly degrades performance. See Figure 3.6 for a qualitative overview.

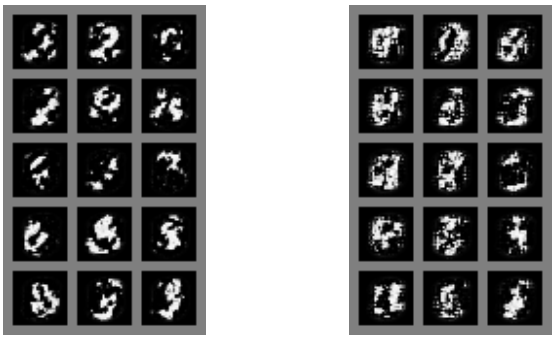
3.7 Combined simulation of noise and limited parameter range

We combined noise and magnitude constraints together to see how they interact. We explored constraint space around reasonable noise and magnitude values and looked at how they affect NLL (Figure 3.7a). The two constraints appear to work well together. In fact, a model with higher noise and small parameter values performs nearly as well as a standard RBMs. We think that the constraint on parameter values may actually be helpful, because they force the RBMs to find good weight vector directions that generalize well, rather than just scaling up its weights



(a) 0.10 **(b)** 0.80 **(c)** 0.99

Figure 3.8 – Random samples after 100,000 Gibbs steps for three different proportions of removed connections.



(a) **(b)**

Figure 3.9 – Random samples after 100,000 Gibbs steps for two RBMs trained with a chimera connectivity pattern, using (3.9a) *pixel blocks* and (3.9b) *extended pixel blocks* pixel-to-units mappings.

to overpower the noise. As always, one should be careful about generalizing these conclusions to values outside the ranges evaluated in these experiments.

3.8 Simulating limited connectivity

We trained RBM by forcing a random subset of weights to be zero and observed how it affected test NLL (Figure 3.7b). It turns out the RBM can cope with a reasonable amount of removed connections: even when half the weights are forced to be zero, test NLL only increases by about 4.3%. However, physical implementations will likely have sparse connectivity; for instance, the connectivity

pattern of a D-Wave machine (Figure 3.1) applied to an RBM with 784 visible units and 784 hidden units is so that over 99% of its connections are removed. In the aforementioned experiment, 99% removed connections results in a disappointing 200.3 ± 0.2 test NLL. When looking at samples (Figure 3.8), we observe that the RBM's representative power decreases as we force more weights to be zero, until samples no longer resemble digits.

Fortunately, physical implementations of an RBM will most likely have some kind of structure to their connectivity pattern, so the results we get by forcing a random subset of the weights to be zero are somewhat pessimistic.

When we train an RBM with 784 visible units and 784 hidden units with chimera connectivity pattern, results are much better. There are many ways to map pixels of an image to visible units of the model; we tried two that seemed the most logical (Figure 3.1). The *pixel blocks* mapping lead to a test NLL of 138.2, while the *extended pixel blocks* mapping lead to a test NLL of 160.9. When we look at samples from both RBMs (Figure 3.9), we see that digit structure is much better preserved than when we randomly force the same proportion of weights to be zero, although samples still barely look like digits. In all cases, the limited architecture seems to be the most damaging constraint studied in this work.

3.9 Conclusion

In this work, we have performed a series of simulation experiments to determine the feasibility of implementing an RBM using physical computation. We have evaluated the impact of three barriers to the success of physical computation: noise on the model parameters, limited range on the model parameters, and limited topology of the model.

We have found that noise on the parameters noticeably degrades performance, though this can be mitigated by training using the same sampler in the negative phase as will be used to draw samples at test time. We have found that the limits on the range of the parameters do not significantly impair the performance of the RBM. Finally, and most importantly, we have found that restrictions on the topology of the model can impair the model's performance more than any of the other limitations we consider. While structured sparsity like in the D-Wave Two

system's chimera topology does perform well for the number of connections it has, the overall number of connections is still low enough to cause many difficulties.

Note however that experiments on noisy weights were performed on fully-connected RBMs. If, as suggested when discussing Figure 3.2, the effect of noisy parameters is dominated by noise on weights because there are more weights than biases, then a constrained architecture might mitigate the effect of noisy weights simply by reducing their number. This needs to be verified in future experiments.

This suggests that quantum hardware designers should concentrate their efforts on reducing noise on weights and on increasing the number of connections between elements in the quantum computers, and quantum machine learning researchers should focus their efforts on designing approaches that can cope with noisy weights and restricted topology.

4 Prologue to Second Article

4.1 Article Details

Adversarially Learned Inference.

Vincent Dumoulin, Ishmael Belghazi, Ben Poole, Olivier Mastropietro, Alex Lamb, Martin Arjovsky, and Aaron Courville. *Proceedings of the International Conference on Learning Representations, 2017.*

Personal Contribution.

I am first contributor to this work with regards to experiment design, analysis and writing.

4.2 Context

Generative adversarial networks (GANs) (Goodfellow et al., 2014) have established themselves as a powerful generative model framework, yet they sidestep the inference problem.

At the time this work was published, there existed no principled adversarial framework for learning inference and generation jointly. This made it difficult to reuse GANs for auxiliary tasks that require inference, such as semi-supervised learning, inpainting, and image manipulation at an abstract level.

4.3 Contributions

We introduce the adversarially learned inference (ALI) model, which jointly learns a generation network and an inference network using an adversarial process.

The generation network maps samples from stochastic latent variables to the data space while the inference network maps training examples in data space to the space of latent variables. An adversarial game is cast between these two networks and a discriminative network is trained to distinguish between joint latent/data-space samples from the generative network and joint samples from the inference network. We illustrate the ability of the model to learn mutually coherent inference and generation networks through the inspection of model samples and reconstructions and confirm the usefulness of the learned representations by obtaining a performance competitive with state-of-the-art on the semi-supervised SVHN and CIFAR10 tasks.

Our paper was published concurrently to a paper titled *Adversarial Feature Learning* (Donahue et al., 2017) which proposes the same adversarial inference framework under the name BiGAN. The two papers differ in their quantitative evaluation setups — BiGAN is instead evaluated on the transferability of its learned features to the ImageNet classification task as well as the Pascal VOC classification, detection, and segmentation tasks — and by the fact that in contrast to BiGAN's deterministic inference network, ALI considers stochastic — and also sometimes implicit — inference networks.

4.4 Recent Developments

As of this writing, generative adversarial networks remain a very active area of research, and as such it is difficult to condense all recent developments in this section. For a more thorough overview, see Goodfellow (2016), Warde-Farley and Goodfellow (2016), and Creswell et al. (2018).

This paper inscribes itself in a rich line of work on implicit inference networks — which promise the benefit of modeling arbitrarily complex posterior distributions without having to maintain a tractable density — that continues to this day. Most related to ALI is work that leverages adversarial training to train implicit inference networks $q(\mathbf{z} \mid \mathbf{x})$ to perform variational inference in generative models that factorize as $p(\mathbf{z})p(\mathbf{x} \mid \mathbf{z})$ — in analogy to recognition networks in variational autoencoders. Makhzani et al. (2015) propose adversarial autoencoders, which decompose the Kullback-Leibler (KL) term of the evidence lower-bound (ELBO)

optimized by a variational autoencoder into an entropy term and a cross-entropy term and minimize the cross-entropy term indirectly via adversarial training, which opens the door to the use of implicit approximate posteriors. However, to the best knowledge of the author of this thesis, this possibility is only mentioned textually and not put into practice. The ALI model proposed in this paper also allows the use of implicit inference networks, this time in a fully adversarial training setup, which is briefly mentioned in the text and put into practice in the toy experiment on 2D gaussian mixtures. This detail is sadly not pointed out in the paper, but is visible in the open-sourced experimental code. The idea of a black-box inference network is centrally featured in work by Mescheder et al. (2017) on adversarial variational Bayes. The authors replace the KL term of the ELBO with a discriminator that is trained to distinguish between samples of the $q(\mathbf{x})q(\mathbf{z} \mid \mathbf{x})$ joint and the $q(\mathbf{x})p(\mathbf{z})$ joint. Similar ideas were also concurrently discussed in Huszár (2017) and in a blog post by the same author. More recently, Makhzani (2018) proposes a fully-adversarial alternative to ALI which leverages two discriminators. The first discriminator is trained to distinguish between samples of the $q(\mathbf{z})p(\mathbf{x} \mid \mathbf{z})$ joint and samples of the $p(\mathbf{z})p(\mathbf{x} \mid \mathbf{z})$ joint, and the second discriminator is trained to distinguish between samples of the $q(\mathbf{z})$ joint and samples of the $p(\mathbf{z})$ joint.

Recent work also explores the use implicit inference networks in a broader context. Belghazi et al. (2018b) extend ALI to accommodate a hierarchy of latent variables. Karaletsos (2016) proposes a local message passing variational inference algorithm for structured models that relies on adversarial training — more specifically, multiple local discriminators are used rather than a single, global discriminator trained on the joint distribution. Tran et al. (2017) introduce hierarchical implicit models (HIMs) along with a likelihood-free variational inference procedure — which employs an implicit variational family for the local latent variables — to perform variational inference on HIMs. Semi-implicit (Yin and Zhou, 2018) and implicit (Titsias and Ruiz, 2018) inference networks are also used to directly optimize the ELBO (or a lower-bound of the ELBO) in a variational inference setting.

Aside from the inference problem in GANs, many questions are yet to be answered satisfyingly, namely guaranteeing a consistent convergence to an equilibrium point of the value function under the min-max optimization procedure, addressing the mode collapse problem, and providing an objective measure of performance. On the adversarial inference front, Li et al. (2017a) outlined a non-identifiability

issue with ALI caused by the lack of cycle-consistency and which can result in poor reconstructions in practice. As a remedy, the authors propose to augment the ALI loss function with a conditional entropy term.

Recently, various flavors of discriminator gradient penalties have emerged as reliable ways to stabilize the training of GANs. These include the gradient penalty proposed by Gulrajani et al. (2017) in the context of Wasserstein GAN training, the gradient penalty proposed by Kodali et al. (2017) in the context of training the non-saturating formulation of GANs, as well as the gradient penalty proposed by Roth et al. (2017) in the context of f-divergence-based GAN training. In fact, Fedus et al. (2018) argues that the gradient penalty itself is generally useful irrespective of the specific GAN-flavoured cost function.

A recent and promising solution to address the mode collapse problem is MINE, an adversarial estimator of mutual information proposed by Belghazi et al. (2018a). The authors use their estimator to encourage the maximization of mutual information between model samples and their latent code, which acts as a proxy to maximize the entropy of the samples.

Finally, the performance measure front, Wu et al. (2017) recently proposed to use annealed importance sampling (AIS) to measure the log-likelihood of decoder-based models, which GANs are an instance of. Heusel et al. (2017) also recently introduced the Fréchet Inception Distance as a better replacement for the widely-used Inception Score (Salimans et al., 2016), which Lucic et al. (2017) leverages along with precision, recall, and F_1 metrics to conduct a large-scale evaluation of existing GAN variants.

5 Adversarially Learned Inference

5.1 Introduction

Deep directed generative model has emerged as a powerful framework for modeling complex high-dimensional datasets. These models permit fast ancestral sampling, but are often challenging to learn due to the complexities of inference. Recently, three classes of algorithms have emerged as effective for learning deep directed generative models: 1) techniques based on the Variational Autoencoder (VAE) that aim to improve the quality and efficiency of inference by learning an inference machine (Kingma and Welling, 2014; Rezende et al., 2014), 2) techniques based on Generative Adversarial Networks (GANs) that bypass inference altogether (Goodfellow et al., 2014) and 3) autoregressive approaches (van den Oord et al., 2016b,c,a) that forego latent representations and instead model the relationship between input variables directly. While all techniques are provably consistent given infinite capacity and data, in practice they learn very different kinds of generative models on typical datasets.

VAE-based techniques learn an approximate inference mechanism that allows reuse for various auxiliary tasks, such as semi-supervised learning or inpainting. They do however suffer from a well-recognized issue of the maximum likelihood training paradigm when combined with a conditional independence assumption on the output given the latent variables: they tend to distribute probability mass diffusely over the data space (Theis et al., 2016). The direct consequence of this is that image samples from VAE-trained models tend to be blurry (Goodfellow et al., 2014; Larsen et al., 2016). Autoregressive models produce outstanding samples but do so at the cost of slow sampling speed and foregoing the learning of an abstract representation of the data. GAN-based approaches represent a good compromise: they learn a generative model that produces higher-quality samples than the best VAE techniques (Radford et al., 2016; Larsen et al., 2016) without sacrificing sam-

pling speed and also make use of a latent representation in the generation process. However, GANs lack an efficient inference mechanism, which prevents them from reasoning about data at an abstract level. For instance, GANs don't allow the sort of neural photo manipulations showcased in (Brock et al., 2017). Recently, efforts have aimed to bridge the gap between VAEs and GANs, to learn generative models with higher-quality samples while learning an efficient inference network (Larsen et al., 2016; Lamb et al., 2016; Dosovitskiy and Brox, 2016). While this is certainly a promising research direction, VAE-GAN hybrids tend to manifest a compromise of the strengths and weaknesses of both approaches.

In this work, we propose a novel approach to integrate efficient inference within the GAN framework. Our approach, called Adversarially Learned Inference (ALI), casts the learning of both an inference machine (or encoder) and a deep directed generative model (or decoder) in an GAN-like adversarial framework. A discriminator is trained to discriminate joint samples of the data and the corresponding latent variable from the encoder (or approximate posterior) from joint samples from the decoder while in opposition, the encoder and the decoder are trained together to fool the discriminator. Not only are we asking the discriminator to distinguish synthetic samples from real data, but we are requiring it to distinguish between two joint distributions over the data space and the latent variables.

With experiments on the Street View House Numbers (SVHN) dataset (Netzer et al., 2011), the CIFAR-10 object recognition dataset (Krizhevsky and Hinton, 2009), the CelebA face dataset (Liu et al., 2015) and a downsampled version of the ImageNet dataset (Russakovsky et al., 2015), we show qualitatively that we maintain the high sample fidelity associated with the GAN framework, while gaining the ability to perform efficient inference. We show that the learned representation is useful for auxiliary tasks by achieving results competitive with the state-of-the-art on the semi-supervised SVHN and CIFAR10 tasks.

5.2 Adversarially learned inference

Consider the two following probability distributions over $\boldsymbol{x}$ and $\boldsymbol{z}$:

— the *encoder* joint distribution $q(\boldsymbol{x}, \boldsymbol{z}) = q(\boldsymbol{x})q(\boldsymbol{z} \mid \boldsymbol{x})$,

— the *decoder* joint distribution $p(\boldsymbol{x}, \boldsymbol{z}) = p(\boldsymbol{z})p(\boldsymbol{x} \mid \boldsymbol{z})$.

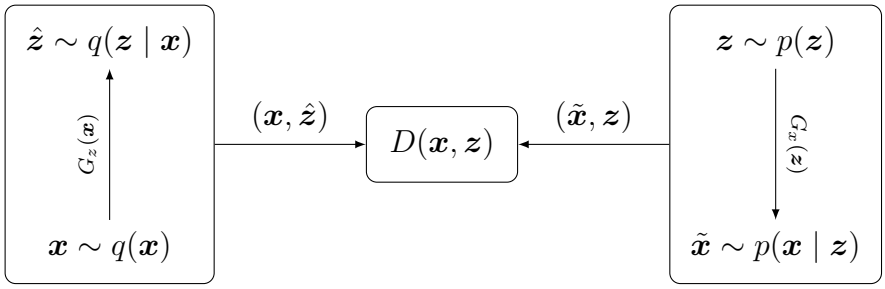


Figure 5.1 – The adversarially learned inference (ALI) game.

These two distributions have marginals that are known to us: the encoder marginal $q(\boldsymbol{x})$ is the empirical data distribution and the decoder marginal $p(\boldsymbol{z})$ is usually defined to be a simple, factorized distribution, such as the standard Normal distribution $p(\boldsymbol{z}) = \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$. As such, the generative process between $q(\boldsymbol{x}, \boldsymbol{z})$ and $p(\boldsymbol{x}, \boldsymbol{z})$ is reversed.

ALI's objective is to match the two joint distributions. If this is achieved, then we are ensured that all marginals match and all conditional distributions also match. In particular, we are assured that the conditional $q(\boldsymbol{z} \mid \boldsymbol{x})$ matches the posterior $p(\boldsymbol{z} \mid \boldsymbol{x})$.

In order to match the joint distributions, an adversarial game is played. Joint pairs $(\boldsymbol{x}, \boldsymbol{z})$ are drawn either from $q(\boldsymbol{x}, \boldsymbol{z})$ or $p(\boldsymbol{x}, \boldsymbol{z})$, and a discriminator network learns to discriminate between the two, while the encoder and decoder networks are trained to fool the discriminator.

The value function describing the game is given by:

$$\begin{aligned} \min_G \max_D V(D, G) &= \mathbb{E}_{q(\boldsymbol{x})}[\log(D(\boldsymbol{x}, G_z(\boldsymbol{x})))] + \mathbb{E}_{p(\boldsymbol{z})}[\log(1 - D(G_x(\boldsymbol{z}), \boldsymbol{z}))] \\ &= \iint q(\boldsymbol{x}) q(\boldsymbol{z} \mid \boldsymbol{x}) \log(D(\boldsymbol{x}, \boldsymbol{z})) d\boldsymbol{x} d\boldsymbol{z} \\ &+ \iint p(\boldsymbol{z}) p(\boldsymbol{x} \mid \boldsymbol{z}) \log(1 - D(\boldsymbol{x}, \boldsymbol{z})) d\boldsymbol{x} d\boldsymbol{z}. \end{aligned} \tag{5.1}$$

An attractive property of adversarial approaches is that they do not require that the conditional densities can be computed; they only require that they can be sampled from in a way that allows gradient backpropagation. In the case of ALI, this means that gradients should propagate from the discriminator network to the encoder and decoder networks.

This can be done using the the reparametrization trick (Kingma, 2013; Bengio et al., 2014, 2013). Instead of sampling directly from the desired distribution, the random variable is computed as a deterministic transformation of some noise such that its distribution is the desired distribution. For instance, if $q(z \mid x) = \mathcal{N}(\mu(x), \sigma^2(x)I)$, one can draw samples by computing

$$z = \mu(x) + \sigma(x) \odot \epsilon, \quad \epsilon \sim \mathcal{N}(0, I). \tag{5.2}$$

More generally, one can employ a change of variable of the form

$$v = f(u, \epsilon) \tag{5.3}$$

where ϵ is some random source of noise.

The discriminator is trained to distinguish between samples from the encoder $(\boldsymbol{x}, \hat{\boldsymbol{z}}) \sim q(\boldsymbol{x}, \boldsymbol{z})$ and samples from the decoder $(\tilde{\boldsymbol{x}}, \boldsymbol{z}) \sim p(\boldsymbol{x}, \boldsymbol{z})$. The generator is trained to fool the discriminator, i.e., to generate $\boldsymbol{x}, \boldsymbol{z}$ pairs from $q(\boldsymbol{x}, \boldsymbol{z})$ or $p(\boldsymbol{x}, \boldsymbol{z})$ that are indistinguishable one from another. See Figure 5.1 for a diagram of the adversarial game and Algorithm 1 for an algorithmic description of the procedure.

In such a setting, and under the assumption of an optimal discriminator, the generator minimizes the Jensen-Shannon divergence (Lin, 1991) between $q(\boldsymbol{x}, \boldsymbol{z})$ and $p(\boldsymbol{x}, \boldsymbol{z})$. This can be shown using the same proof sketch as in the original GAN paper (Goodfellow et al., 2014).

5.2.1 Relation to GAN

ALI bears close resemblance to GAN, but it differs from it in the two following ways:

— The generator has two components: the encoder, $G_z(\boldsymbol{x})$, which maps data samples $\boldsymbol{x}$ to $\boldsymbol{z}$-space, and the decoder $G_x(\boldsymbol{z})$, which maps samples from the prior $p(\boldsymbol{z})$ (a source of noise) to the input space.

— The discriminator is trained to distinguish between joint pairs $(\boldsymbol{x}, \hat{\boldsymbol{z}} = G_x(\boldsymbol{x}))$ and $(\tilde{\boldsymbol{x}} = G_x(\boldsymbol{z}), \boldsymbol{z})$, as opposed to marginal samples $\boldsymbol{x} \sim q(\boldsymbol{x})$ and $\tilde{\boldsymbol{x}} \sim p(\boldsymbol{x})$.

Algorithm 1 The ALI training procedure.

$\theta_g, \theta_d \leftarrow$ initialize network parameters
repeat
  $\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(M)} \sim q(\boldsymbol{x})$ ▷ Draw M samples from the dataset and the prior
  $\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(M)} \sim p(\boldsymbol{z})$
  $\hat{\boldsymbol{z}}^{(i)} \sim q(\boldsymbol{z} \mid \boldsymbol{x} = \boldsymbol{x}^{(i)}), \quad i = 1, \ldots, M$ ▷ Sample from the conditionals
  $\tilde{\boldsymbol{x}}^{(j)} \sim p(\boldsymbol{x} \mid \boldsymbol{z} = \boldsymbol{z}^{(j)}), \quad j = 1, \ldots, M$
  $\rho_q^{(i)} \leftarrow D(\boldsymbol{x}^{(i)}, \hat{\boldsymbol{z}}^{(i)}), \quad i = 1, \ldots, M$ ▷ Compute discriminator predictions
  $\rho_p^{(j)} \leftarrow D(\tilde{\boldsymbol{x}}^{(j)}, \boldsymbol{z}^{(j)}), \quad j = 1, \ldots, M$
  $\mathcal{L}_d \leftarrow -\frac{1}{M}\sum_{i=1}^{M} \log(\rho_q^{(i)}) - \frac{1}{M}\sum_{j=1}^{M} log(1 - \rho_p^{(j)})$ ▷ Compute discriminator loss
  $\mathcal{L}_g \leftarrow -\frac{1}{M}\sum_{i=1}^{M} \log(1 - \rho_q^{(i)}) - \frac{1}{M}\sum_{j=1}^{M} \log(\rho_p^{(j)})$ ▷ Compute generator loss
  $\theta_d \leftarrow \theta_d - \nabla_{\theta_d}\mathcal{L}_d$ ▷ Gradient update on discriminator network
  $\theta_g \leftarrow \theta_g - \nabla_{\theta_g}\mathcal{L}_g$ ▷ Gradient update on generator networks
until convergence

5.2.2 Alternative approaches to feedforward inference in GAN

The ALI training procedure is not the only way one could learn a feedforward inference network in a GAN setting.

In recent work, Chen et al. (2016) introduce a model called InfoGAN which minimizes the mutual information between a subset $\boldsymbol{c}$ of the latent code and $\boldsymbol{x}$ through the use of an auxiliary distribution $Q(\boldsymbol{c} \mid \boldsymbol{x})$. However, this does not correspond to full inference on $\boldsymbol{z}$, as only the value for $\boldsymbol{c}$ is inferred. Additionally, InfoGAN requires that $Q(\mathbf{c} \mid \mathbf{x})$ is a tractable approximate posterior that can be sampled from and evaluated. ALI only requires that inference networks can be sampled from, allowing it to represent arbitrarily complex posterior distributions.

One could learn the inverse mapping from GAN samples: this corresponds to learning an encoder to reconstruct $\boldsymbol{z}$, i.e. finding an encoder such that $\mathbb{E}_{z \sim p(z)}[\|z - G_z(G_x(z))\|_2^2] \approx 0$. We are not aware of any work that reports results for this approach. This resembles the InfoGAN learning procedure but with a fixed generative model and a factorial Gaussian posterior with a fixed diagonal variance.

Alternatively, one could decompose training into two phases. In the first phase, a GAN is trained normally. In the second phase, the GAN's decoder is frozen and an encoder is trained following the ALI procedure (i.e., a discriminator taking *both*

$\boldsymbol{x}$ and $\boldsymbol{z}$ as input is introduced). We call this *post-hoc learned inference*. In this setting, the encoder and the decoder cannot interact together during training and the encoder must work with whatever the decoder has learned during GAN training. Post-hoc learned inference may be suboptimal if this interaction is beneficial to modeling the data distribution.

5.2.3 Generator value function

As with GAN, when GAN's discriminator gets too far ahead, its generator may have a hard time minimizing the value function in Equation 5.1. If the discriminator's output is sigmoidal, then the gradient of the value function with respect to the discriminator's output vanishes to zero as the output saturates.

As a workaround, the generator is trained to maximize

$$V'(D, G) = \mathbb{E}_{q(\boldsymbol{x})}[\log(1 - D(\boldsymbol{x}, G_{\boldsymbol{z}}(\boldsymbol{x})))] + \mathbb{E}_{p(\boldsymbol{z})}[\log(D(G_{\boldsymbol{x}}(\boldsymbol{z}), \boldsymbol{z}))] \tag{5.4}$$

which has the same fixed points but whose gradient is stronger when the discriminator's output saturates.

The adversarial game does not require an analytical expression for the joint distributions. This means we can introduce variable changes without having to know the explicit distribution over the new variable. For instance, sampling from $p(z)$ could be done by sampling $\epsilon \sim \mathcal{N}(0, I)$ and passing it through an arbitrary differentiable function $z = f(\epsilon)$.

However, gradient propagation into the encoder and decoder networks relies on the reparametrization trick, which means that ALI is not directly applicable to either applications with discrete data or to models with discrete latent variables.

5.2.4 Discriminator optimality

Proposition 1. *Given a fixed generator G, the optimal discriminator is given by*

$$D^*(x, z) = \frac{q(x, z)}{q(x, z) + p(x, z)}. \tag{5.5}$$

(a) SVHN samples.

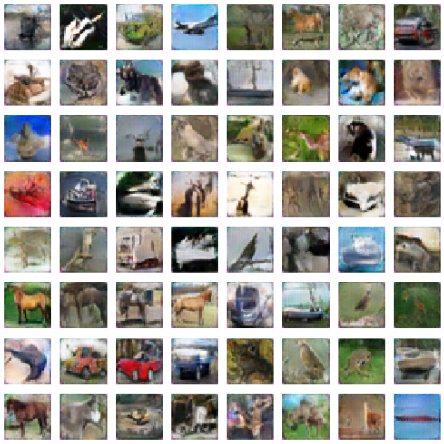
(b) SVHN reconstructions.

Figure 5.2 – Samples and reconstructions on the SVHN dataset. For the reconstructions, odd columns are original samples from the validation set and even columns are corresponding reconstructions (e.g., second column contains reconstructions of the first column's validation set samples).

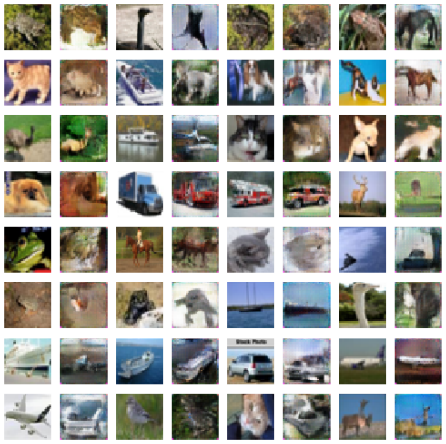
(a) CelebA samples.

(b) CelebA reconstructions.

Figure 5.3 – Samples and reconstructions on the CelebA dataset. For the reconstructions, odd columns are original samples from the validation set and even columns are corresponding reconstructions.

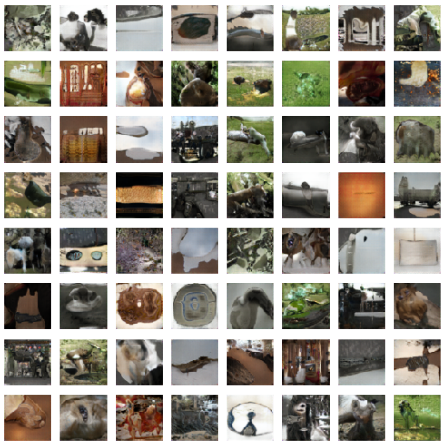


(a) CIFAR10 samples.

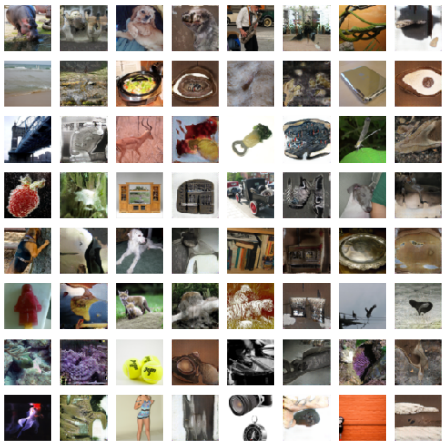


(b) CIFAR10 reconstructions.

Figure 5.4 – Samples and reconstructions on the CIFAR10 dataset. For the reconstructions, odd columns are original samples from the validation set and even columns are corresponding reconstructions.



(a) Tiny ImageNet samples.



(b) Tiny ImageNet reconstructions.

Figure 5.5 – Samples and reconstructions on the Tiny ImageNet dataset. For the reconstructions, odd columns are original samples from the validation set and even columns are corresponding reconstructions.

Proof. For a fixed generator G, the complete data value function is

$$V(D,G) = \mathbb{E}_{x,z\sim q(x,z)}[\log(D(x,z))] + \mathbb{E}_{x,z\sim p(x,z)}[\log(1-D(x,z))]. \tag{5.6}$$

The result follows by the concavity of the log and the simplified Euler-Lagrange equation first order conditions on $(x,z) \to D(x,z)$. □

5.2.5 Relationship with the Jensen-Shannon divergence

Proposition 2. *Under an optimal discriminator D^*, the generator minimizes the Jensen-Shanon divergence which attains its minimum if and only if $q(\boldsymbol{x},\boldsymbol{z}) = p(\boldsymbol{x},\boldsymbol{z})$.*

Proof. The proof is a straightforward extension of the proof in Goodfellow et al. (2014). □

5.2.6 Invertibility

Proposition 3. *Assuming optimal discriminator D and generator G. If the encoder $G_{\boldsymbol{x}}$ is deterministic, then $G_{\boldsymbol{x}} = G_{\boldsymbol{z}}^{-1}$ and $G_{\boldsymbol{z}} = G_{\boldsymbol{x}}^{-1}$ almost everywhere.*

Sketch of proof. Consider the event $R_\epsilon = \{\boldsymbol{x} : \|x - (G_{\boldsymbol{x}} \circ G_{\boldsymbol{z}})(\boldsymbol{x}))\| > \epsilon\}$ for some positive ϵ. This set can be seen as a section of the $(\boldsymbol{x},\boldsymbol{z})$ space over the elements $\boldsymbol{z}$ such that $\boldsymbol{z} = G_{\boldsymbol{z}}(x)$. The generator being optimal, the probabilities of R_ϵ under $p(\boldsymbol{x},\boldsymbol{z})$ and $q(\boldsymbol{x},\boldsymbol{z})$ are equal. Now $p(\boldsymbol{x} \mid \boldsymbol{z}) = \delta_{x-G_x(z)}$, where δ is the Dirac delta distribution. This is enough to show that there are no x satisfying the event R_ϵ and thus $G_{\boldsymbol{x}} = G_z^{-1}$ almost everywhere. By symmetry, the same argument can be applied to show that $G_{\boldsymbol{z}} = G_{\boldsymbol{x}}^{-1}$.
The complete proof is given in (Donahue et al., 2017), in which the authors independently examine the same model structure under the name Bidirectional GAN (BiGAN). □

5.3 Related Work

Other recent papers explore hybrid approaches to generative modeling. One such approach is to relax the probabilistic interpretation of the VAE model by replacing either the KL-divergence term or the reconstruction term with variants that have better properties. The adversarial autoencoder model (Makhzani et al., 2015) replaces the KL-divergence term with a discriminator that is trained to distinguish between approximate posterior and prior samples, which provides a more flexible approach to matching the marginal $q(\boldsymbol{z})$ and the prior. Other papers explore replacing the reconstruction term with either GANs or auxiliary networks. Larsen et al. (2016) collapse the decoder of a VAE and the generator of a GAN into one network in order to supplement the reconstruction loss with a learned similarity metric. Lamb et al. (2016) use the hidden layers of a pre-trained classifier as auxiliary reconstruction losses to help the VAE focus on higher-level details when reconstructing. Dosovitskiy and Brox (2016) combine both ideas into a unified loss function.

ALI's approach is also reminiscent of the adversarial autoencoder model, which employs a GAN to distinguish between samples from the approximate posterior distribution $q(\boldsymbol{z} \mid \boldsymbol{x})$ and prior samples. However, unlike adversarial autoencoders, no explicit reconstruction loss is being optimized in ALI, and the discriminator receives joint pairs of samples $(\boldsymbol{x}, \boldsymbol{z})$ rather than marginal $\boldsymbol{z}$ samples.

Independent work by Donahue et al. (2017) proposes the same model under the name Bidirectional GAN (BiGAN), in which the authors emphasize the learned features' usefulness for auxiliary supervised and semi-supervised tasks. The main difference in terms of experimental setting is that they use a deterministic $q(\boldsymbol{z} \mid \boldsymbol{x})$ network, whereas we use a stochastic network. In our experience, this does not make a big difference when $\boldsymbol{x}$ is a deterministic function of $\boldsymbol{z}$ as the stochastic inference networks tend to become determinstic as training progresses. When using stochastic mappings from $\boldsymbol{z}$ to $\boldsymbol{x}$, the additional flexiblity of stochastic posteriors is critical.

5.4 Experimental results

We applied ALI to four different datasets, namely CIFAR10 (Krizhevsky and Hinton, 2009), SVHN (Netzer et al., 2011), CelebA (Liu et al., 2015) and a center-cropped, 64×64 version of the ImageNet dataset (Russakovsky et al., 2015). [i]

Transposed convolutions are used in $G_x(\boldsymbol{z})$. This operation corresponds to the transpose of the matrix representation of a convolution, i.e., the gradient of the convolution with respect to its inputs. For more details about transposed convolutions and related operations, see Dumoulin and Visin (2016); Shi et al. (2016); Odena et al. (2016).

5.4.1 Samples and Reconstructions

For each dataset, samples are presented (Figures 5.2a, 5.3a 5.4a and 5.5a). They exhibit the same image fidelity as samples from other adversarially-trained models.

We also qualitatively evaluate the fit between the conditional distribution $q(\boldsymbol{z} \mid \boldsymbol{x})$ and the posterior distribution $p(\boldsymbol{z} \mid \boldsymbol{x})$ by sampling $\hat{\boldsymbol{z}} \sim q(\boldsymbol{z} \mid \boldsymbol{x})$ and $\hat{\boldsymbol{x}} \sim p(\boldsymbol{x} \mid \boldsymbol{z} = \hat{\boldsymbol{z}})$ (Figures 5.2b, 5.3b, 5.4b and 5.5b). This corresponds to reconstructing the input in a VAE setting. Note that the ALI training objective does *not* involve an explicit reconstruction loss.

We observe that reconstructions are not always faithful reproductions of the inputs. They retain the same crispness and quality characteristic to adversarially-trained models, but oftentimes make mistakes in capturing exact object placement, color, style and (in extreme cases) object identity. The extent to which reconstructions deviate from the inputs varies between datasets: on CIFAR10, which arguably constitutes a more complex input distribution, the model exhibits less faithful reconstructions. This leads us to believe that poor reconstructions are a sign of underfitting.

This failure mode represents an interesting departure from the bluriness characteristic to the typical VAE setup. We conjecture that in the underfitting regime, the latent variable representation learned by ALI is potentially more invariant to less interesting factors of variation in the input and do not devote model capacity

i. The code for all experiments can be found at https://github.com/IshmaelBelghazi/ALI. Readers can also consult the accompanying website at https://ishmaelbelghazi.github.io/ALI.

Model	Misclassification rate
VAE (M1 + M2) [i]	36.02
SWWAE with dropout [ii]	23.56
DCGAN + L2-SVM [iii]	22.18
SDGM [iv]	16.61
GAN (feature matching) [v]	$\mathbf{8.11 \pm 1.3}$
ALI (ours, L2-SVM)	19.14 ± 0.50
ALI (ours, no feature matching)	**7.3**

Table 5.1 – SVHN test set misclassification rate

.

to capturing these factors.

5.4.2 Latent space interpolations

As a sanity check for overfitting, we look at latent space interpolations between validation set examples (Figure 5.6). We sample pairs of validation set examples $\mathbf{x}_1$ and $\mathbf{x}_2$ and project them into $\mathbf{z}_1$ and $\mathbf{z}_2$ by sampling from the encoder. We then linearly interpolate between $\mathbf{z}_1$ and $\mathbf{z}_2$ and pass the intermediary points through the decoder to plot the input-space interpolations.

We observe smooth transitions between pairs of examples, and intermediary images remain believable. This is an indicator that ALI is not concentrating its probability mass exclusively around training examples, but rather has learned latent features that generalize well.

5.4.3 Semi-supervised learning

We investigate the usefulness of the latent representation learned by ALI through semi-supervised benchmarks on SVHN and CIFAR10.

i. Kingma et al. (2014)
ii. Zhao et al. (2016)
iii. Radford et al. (2016)
iv. Maaløe et al. (2016)
v. Salimans et al. (2016)

Figure 5.6 – Latent space interpolations on the CelebA validation set. Left and right columns correspond to the original pairs $\mathbf{x}_1$ and $\mathbf{x}_2$, and the columns in between correspond to the decoding of latent representations interpolated linearly from $\mathbf{z}_1$ to $\mathbf{z}_2$. Unlike other adversarial approaches like DCGAN (Radford et al., 2016), ALI allows one to interpolate between actual data points.

We first compare with GAN on SVHN by following the procedure outlined in Radford et al. (2016). We train an L2-SVM on the learned representations of a model trained on SVHN. The last three hidden layers of the encoder as well as its output are concatenated to form a 8960-dimensional feature vector. A 10,000 example held-out validation set is taken from the training set and is used for model selection. The SVM is trained on 1000 examples taken at random from the remainder of the training set. The test error rate is measured for 100 different SVMs trained on different random 1000-example training sets, and the average error rate is measured along with its standard deviation.

Using ALI's inference network as opposed to the discriminator to extract features, we achieve a misclassification rate that is roughly $3.00 \pm 0.50\%$ lower than reported in Radford et al. (2016) (Table 5.1), which suggests that ALI's inference mechanism is beneficial to the semi-supervised learning task.

We then investigate ALI's performance when label information is taken into account during training. We adapt the discriminative model proposed in Salimans et al. (2016). The discriminator takes x and z as input and outputs a distribution over $K + 1$ classes, where K is the number of categories. When label information is available for $q(x, z)$ samples, the discriminator is expected to predict the label. When no label information is available, the discriminator is expected to predict $K + 1$ for $p(x, z)$ samples and $k \in \{1, \ldots, K\}$ for $q(x, z)$ samples.

Interestingly, Salimans et al. (2016) found that they required an alternative training strategy for the generator where it tries to match first-order statistics in

Number of labeled examples	1000	2000	4000	8000
Model	Misclassification rate			
Ladder network [i]			20.40	
CatGAN [ii]			19.58	
GAN (feat. matching) [iii]	**21.83±2.01**	**19.61±2.09**	**18.63±2.32**	**17.72±1.82**
ALI (ours, no feat. matching)	**19.98±0.89**	**19.09±0.44**	**17.99±1.62**	**17.05±1.49**

Table 5.2 – CIFAR10 test set misclassification rate for semi-supervised learning using different numbers of trained labeled examples. For ALI, error bars correspond to 3 times the standard deviation.

the discriminator's intermediate activations with respect to the data distribution (they refer to this as *feature matching*). We found that ALI did not require feature matching to obtain comparable results. We achieve results competitive with the state-of-the-art, as shown in Tables 5.1 and 5.2. Table 5.2 shows that ALI offers a modest improvement over Salimans et al. (2016), more specifically for 1000 and 2000 labeled examples.

We are still investigating the differences between ALI and GAN with respect to feature matching, but we conjecture that the latent representation learned by ALI is better untangled with respect to the classification task and that it generalizes better.

5.4.4 Conditional Generation

We extend ALI to match a conditional distribution. Let $\boldsymbol{y}$ represent a fully observed conditioning variable. In this setting, the value function reads

$$\begin{aligned} V(D,G) =& \mathbb{E}_{q(\boldsymbol{x})\,p(\boldsymbol{y})}[\log(D(\boldsymbol{x}, G_z(\boldsymbol{x},\boldsymbol{y}),\boldsymbol{y}))] + \\ & \mathbb{E}_{p(\boldsymbol{z})\,p(\boldsymbol{y})}[\log(1 - D(G_x(\boldsymbol{z},\boldsymbol{y}),\boldsymbol{z},\boldsymbol{y}))] \end{aligned} \tag{5.7}$$

We apply the conditional version of ALI to CelebA using the dataset's 40 binary attributes. The attributes are linearly embedded in the encoder, decoder and dis-

i. Rasmus et al. (2015)
ii. Springenberg (2016)
iii. Salimans et al. (2016)

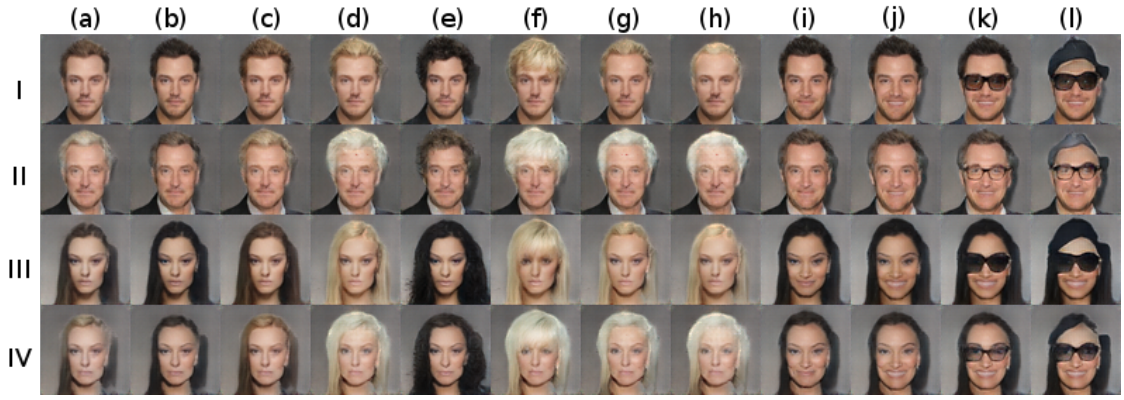


Figure 5.7 – Conditional generation sequence. We sample a single fixed latent code z. Each row has a subset of attributes that are held constant across columns. The attributes are male, attractive, young for row I; male, attractive, older for row II; female, attractive, young for row III; female, attractive, older for Row IV. Attributes are then varied uniformly over rows across all columns in the following sequence: (b) black hair; (c) brown hair; (d) blond hair; (e) black hair, wavy hair; (f) blond hair, bangs; (g) blond hair, receding hairline; (h) blond hair, balding; (i) black hair, smiling; (j) black hair, smiling, mouth slightly open; (k) black hair, smiling, mouth slightly open, eyeglasses; (l) black hair, smiling, mouth slightly open, eyeglasses, wearing hat.

criminator. We observe how a single element of the latent space z changes with respect to variations in the attributes vector y. Conditional samples are shown in Figure 5.7.

5.4.5 Importance of learning inference jointly with generation

To highlight the role of the inference network during learning, we performed an experiment on a toy dataset for which $q(\boldsymbol{x})$ is a 2D gaussian mixture with 25 mixture components laid out on a grid. The covariance matrices and centroids have been chosen such that the distribution exhibits lots of modes separated by large low-probability regions, which makes it a decently hard task despite the 2D nature of the dataset.

We trained ALI and GAN on 100,000 $q(\boldsymbol{x})$ samples. The decoder and discriminator architectures are identical between ALI and GAN (except for the input of the discriminator, which receives the concatenation of $\mathbf{x}$ and $\mathbf{z}$ in the ALI case). Each model was trained 10 times using Adam (Kingma and Ba, 2014) with random learning rate and β_1 values, and the weights were initialized by drawing from

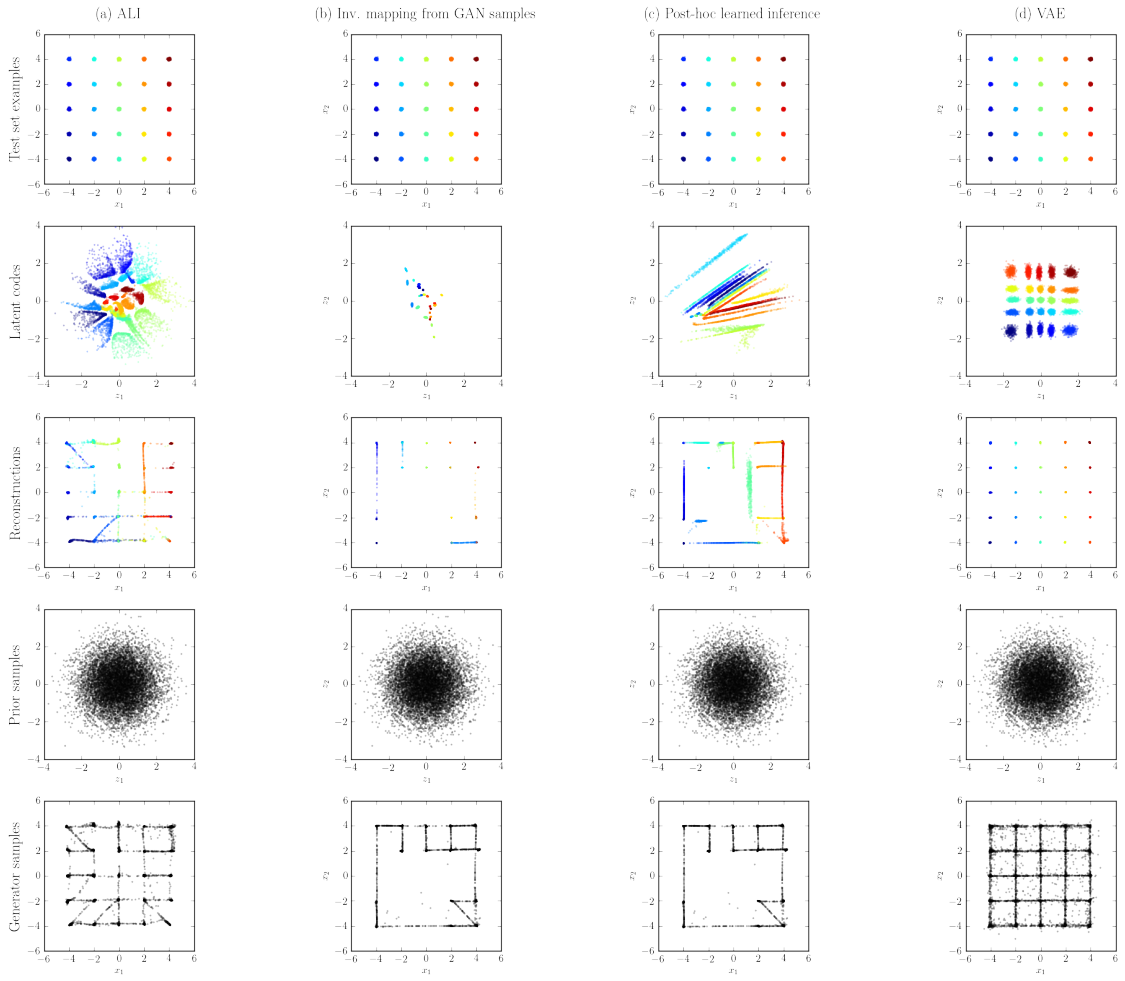


Figure 5.8 – Comparison of (a) ALI, (b) GAN with an encoder learned to reconstruct latent samples (c) GAN with an encoder learned through ALI, (d) variational autoencoder (VAE) on a 2D toy dataset. The ALI model in (a) does a much better job of covering the latent space (second row) and producing good samples than the two GAN models (b, c) augmented with an inference mechanism.

a gaussian distribution with a random standard deviation.

We measured the extent to which the trained models covered all 25 modes by drawing 10,000 samples from their $p(\boldsymbol{x})$ distribution and assigning each sample to a $q(\boldsymbol{x})$ mixture component according to the mixture responsibilities. We defined a dropped mode as one that wasn't assigned to *any* sample. Using this definition, we found that ALI models covered 13.4 ± 5.8 modes on average (min: 8, max: 25) while GAN models covered 10.4 ± 9.2 modes on average (min: 1, max: 22).

We then selected the best-covering ALI and GAN models, and the GAN model was augmented with an encoder using the *learned inverse mapping* and *post-hoc learned inference* procedures outlined in subsection 5.2.2. The encoders learned for GAN inference have the same architecture as ALI's encoder. We also trained a VAE with the same encoder-decoder architecture as ALI to outline the qualitative differences between ALI and VAE models.

We then compared each model's inference capabilities by reconstructing 10,000 held-out samples from $q(\boldsymbol{x})$. Figure 5.8 summarizes the experiment. We observe the following:

- The ALI encoder models a marginal distribution $q(\boldsymbol{z})$ that matches $\boldsymbol{p(z)}$ fairly well (row 2, column a). The learned representation does a decent job at clustering and organizing the different mixture components.
- The GAN generator (row 5, columns b-c) has more trouble reaching all the modes than the ALI generator (row 5, column a), even over 10 runs of hyperparameter search.
- Learning an inverse mapping from GAN samples does not work very well: the encoder has trouble covering the prior marginally and the way it clusters mixture components is not very well organized (row 2, column b). As discussed in subsection 5.2.2, reconstructions suffer from the generator dropping modes.
- Learning inference post-hoc doesn't work as well as training the encoder and the decoder jointly. As had been hinted at in subsection 5.2.2, it appears that adversarial training benefits from learning inference at training time in terms of mode coverage. This also negatively impacts how the latent space is organized (row 2, column c). However, it appears to be better at matching $q(\boldsymbol{z})$ and $p(\boldsymbol{z})$ than when inference is learned through inverse mapping from GAN samples.

— Due to the nature of the loss function being optimized, the VAE model covers all modes easily (row 5, column d) and excels at reconstructing data samples (row 3, column d). However, they have a much more pronounced tendency to smear out their probability density (row 5, column d) and leave "holes" in $\mathbf{q}(\mathbf{z})$ (row 2, column d). Note however that recent approaches such as Inverse Autoregressive Flow (Kingma et al., 2016) may be used to improve on this, at the cost of a more complex mathematical framework.

In summary, this experiment provides evidence that adversarial training benefits from learning an inference mechanism jointly with the decoder. Furthermore, it shows that our proposed approach for learning inference in an adversarial setting is superior to the other approaches investigated.

5.5 Conclusion

We introduced the adversarially learned inference (ALI) model, which jointly learns a generation network and an inference network using an adversarial process. The model learns mutually coherent inference and generation networks, as exhibited by its reconstructions. The induced latent variable mapping is shown to be useful, achieving results competitive with the state-of-the-art on the semi-supervised SVHN and CIFAR10 tasks.

Acknowledgements

The authors would like to acknowledge the support of the following agencies for research funding and computing support: NSERC, Calcul Québec, Compute Canada. We would also like to thank the developers of Theano (Bergstra et al., 2010; Bastien et al., 2012; Theano Development Team, 2016), Blocks and Fuel (van Merriënboer et al., 2015), which were used extensively for this work. Finally, we would like to thank Yoshua Bengio, David Warde-Farley, Yaroslav Ganin and Laurent Dinh for their valuable feedback.

Appendix

Operation	Kernel	Strides	Feature maps	BN?	Dropout	Nonlinearity
$G_z(x)$ – $3 \times 32 \times 32$ input						
Convolution	5×5	1×1	32	√	0.0	Leaky ReLU
Convolution	4×4	2×2	64	√	0.0	Leaky ReLU
Convolution	4×4	1×1	128	√	0.0	Leaky ReLU
Convolution	4×4	2×2	256	√	0.0	Leaky ReLU
Convolution	4×4	1×1	512	√	0.0	Leaky ReLU
Convolution	1×1	1×1	512	√	0.0	Leaky ReLU
Convolution	1×1	1×1	128	×	0.0	Linear
$G_x(z)$ – $64 \times 1 \times 1$ input						
Transposed convolution	4×4	1×1	256	√	0.0	Leaky ReLU
Transposed convolution	4×4	2×2	128	√	0.0	Leaky ReLU
Transposed convolution	4×4	1×1	64	√	0.0	Leaky ReLU
Transposed convolution	4×4	2×2	32	√	0.0	Leaky ReLU
Transposed convolution	5×5	1×1	32	√	0.0	Leaky ReLU
Convolution	1×1	1×1	32	√	0.0	Leaky ReLU
Convolution	1×1	1×1	3	×	0.0	Sigmoid
$D(x)$ – $3 \times 32 \times 32$ input						
Convolution	5×5	1×1	32	×	0.2	Maxout
Convolution	4×4	2×2	64	×	0.5	Maxout
Convolution	4×4	1×1	128	×	0.5	Maxout
Convolution	4×4	2×2	256	×	0.5	Maxout
Convolution	4×4	1×1	512	×	0.5	Maxout
$D(z)$ – $64 \times 1 \times 1$ input						
Convolution	1×1	1×1	512	×	0.2	Maxout
Convolution	1×1	1×1	512	×	0.5	Maxout
$D(x, z)$ – $1024 \times 1 \times 1$ input						
Concatenate $D(x)$ and $D(z)$ along the channel axis						
Convolution	1×1	1×1	1024	×	0.5	Maxout
Convolution	1×1	1×1	1024	×	0.5	Maxout
Convolution	1×1	1×1	1	×	0.5	Sigmoid
Optimizer	Adam ($\alpha = 10^{-4}$, $\beta_1 = 0.5$, $\beta_2 = 10^{-3}$)					
Batch size	100					
Epochs	6475					
Leaky ReLU slope, maxout pieces	0.1, 2					
Weight, bias initialization	Isotropic gaussian ($\mu = 0$, $\sigma = 0.01$), Constant(0)					

Table 5.3 – CIFAR10 model hyperparameters (unsupervised). Maxout layers (Goodfellow et al., 2013b) are used in the discriminator.

Operation	Kernel	Strides	Feature maps	BN?	Dropout	Nonlinearity
$G_z(x)$ – $3 \times 32 \times 32$ input						
Convolution	5×5	1×1	32	$\surd$	0.0	Leaky ReLU
Convolution	4×4	2×2	64	$\surd$	0.0	Leaky ReLU
Convolution	4×4	1×1	128	$\surd$	0.0	Leaky ReLU
Convolution	4×4	2×2	256	$\surd$	0.0	Leaky ReLU
Convolution	4×4	1×1	512	$\surd$	0.0	Leaky ReLU
Convolution	1×1	1×1	512	$\surd$	0.0	Leaky ReLU
Convolution	1×1	1×1	512	$\times$	0.0	Linear
$G_x(z)$ – $256 \times 1 \times 1$ input						
Transposed convolution	4×4	1×1	256	$\surd$	0.0	Leaky ReLU
Transposed convolution	4×4	2×2	128	$\surd$	0.0	Leaky ReLU
Transposed convolution	4×4	1×1	64	$\surd$	0.0	Leaky ReLU
Transposed convolution	4×4	2×2	32	$\surd$	0.0	Leaky ReLU
Transposed convolution	5×5	1×1	32	$\surd$	0.0	Leaky ReLU
Convolution	1×1	1×1	32	$\surd$	0.0	Leaky ReLU
Convolution	1×1	1×1	3	$\times$	0.0	Sigmoid
$D(x)$ – $3 \times 32 \times 32$ input						
Convolution	5×5	1×1	32	$\times$	0.2	Leaky ReLU
Convolution	4×4	2×2	64	$\surd$	0.2	Leaky ReLU
Convolution	4×4	1×1	128	$\surd$	0.2	Leaky ReLU
Convolution	4×4	2×2	256	$\surd$	0.2	Leaky ReLU
Convolution	4×4	1×1	512	$\surd$	0.2	Leaky ReLU
$D(z)$ – $256 \times 1 \times 1$ input						
Convolution	1×1	1×1	512	$\times$	0.2	Leaky ReLU
Convolution	1×1	1×1	512	$\times$	0.2	Leaky ReLU
$D(x, z)$ – $1024 \times 1 \times 1$ input						
Concatenate $D(x)$ and $D(z)$ along the channel axis						
Convolution	1×1	1×1	1024	$\times$	0.2	Leaky ReLU
Convolution	1×1	1×1	1024	$\times$	0.2	Leaky ReLU
Convolution	1×1	1×1	1	$\times$	0.2	Sigmoid

Optimizer	Adam ($\alpha = 10^{-4}$, $\beta_1 = 0.5$, $\beta_2 = 10^{-3}$)
Batch size	100
Epochs	100
Leaky ReLU slope	0.01
Weight, bias initialization	Isotropic gaussian ($\mu = 0$, $\sigma = 0.01$), Constant(0)

Table 5.4 – SVHN model hyperparameters (unsupervised).

Operation	Kernel	Strides	Feature maps	BN?	Dropout	Nonlinearity
$G_z(x)$ – $3 \times 64 \times 64$ input						
Convolution	2×2	1×1	64	$\surd$	0.0	Leaky ReLU
Convolution	7×7	2×2	128	$\surd$	0.0	Leaky ReLU
Convolution	5×5	2×2	256	$\surd$	0.0	Leaky ReLU
Convolution	7×7	2×2	256	$\surd$	0.0	Leaky ReLU
Convolution	4×4	1×1	512	$\surd$	0.0	Leaky ReLU
Convolution	1×1	1×1	512	$\times$	0.0	Linear
$G_x(z)$ – $512 \times 1 \times 1$ input						
Transposed convolution	4×4	1×1	512	$\surd$	0.0	Leaky ReLU
Transposed convolution	7×7	2×2	256	$\surd$	0.0	Leaky ReLU
Transposed convolution	5×5	2×2	256	$\surd$	0.0	Leaky ReLU
Transposed convolution	7×7	2×2	128	$\surd$	0.0	Leaky ReLU
Transposed convolution	2×2	1×1	64	$\surd$	0.0	Leaky ReLU
Convolution	1×1	1×1	3	$\times$	0.0	Sigmoid
$D(x)$ – $3 \times 64 \times 64$ input						
Convolution	2×2	1×1	64	$\surd$	0.0	Leaky ReLU
Convolution	7×7	2×2	128	$\surd$	0.0	Leaky ReLU
Convolution	5×5	2×2	256	$\surd$	0.0	Leaky ReLU
Convolution	7×7	2×2	256	$\surd$	0.0	Leaky ReLU
Convolution	4×4	1×1	512	$\surd$	0.0	Leaky ReLU
$D(z)$ – $512 \times 1 \times 1$ input						
Convolution	1×1	1×1	1024	$\times$	0.2	Leaky ReLU
Convolution	1×1	1×1	1024	$\times$	0.2	Leaky ReLU
$D(x, z)$ – $1536 \times 1 \times 1$ input						
Concatenate $D(x)$ and $D(z)$ along the channel axis						
Convolution	1×1	1×1	2048	$\times$	0.2	Leaky ReLU
Convolution	1×1	1×1	2048	$\times$	0.2	Leaky ReLU
Convolution	1×1	1×1	1	$\times$	0.2	Sigmoid

Optimizer	Adam ($\alpha = 10^{-4}$, $\beta_1 = 0.5$)
Batch size	100
Epochs	123
Leaky ReLU slope	0.02
Weight, bias initialization	Isotropic gaussian ($\mu = 0$, $\sigma = 0.01$), Constant(0)

Table 5.5 – CelebA model hyperparameters (unsupervised).

Operation	Kernel	Strides	Feature maps	BN?	Dropout	Nonlinearity
$G_z(x)$ – $3 \times 64 \times 64$ input						
Convolution	4×4	2×2	64	$\surd$	0.0	Leaky ReLU
Convolution	4×4	1×1	64	$\surd$	0.0	Leaky ReLU
Convolution	4×4	2×2	128	$\surd$	0.0	Leaky ReLU
Convolution	4×4	1×1	128	$\surd$	0.0	Leaky ReLU
Convolution	4×4	2×2	256	$\surd$	0.0	Leaky ReLU
Convolution	4×4	1×1	256	$\surd$	0.0	Leaky ReLU
Convolution	1×1	1×1	2048	$\surd$	0.0	Leaky ReLU
Convolution	1×1	1×1	2048	$\surd$	0.0	Leaky ReLU
Convolution	1×1	1×1	512	$\times$	0.0	Linear
$G_x(z)$ – $256 \times 1 \times 1$ input						
Convolution	1×1	1×1	2048	$\surd$	0.0	Leaky ReLU
Convolution	1×1	1×1	256	$\surd$	0.0	Leaky ReLU
Transposed convolution	4×4	1×1	256	$\surd$	0.0	Leaky ReLU
Transposed convolution	4×4	2×2	128	$\surd$	0.0	Leaky ReLU
Transposed convolution	4×4	1×1	128	$\surd$	0.0	Leaky ReLU
Transposed convolution	4×4	2×2	64	$\surd$	0.0	Leaky ReLU
Transposed convolution	4×4	1×1	64	$\surd$	0.0	Leaky ReLU
Transposed convolution	4×4	2×2	64	$\surd$	0.0	Leaky ReLU
Convolution	1×1	1×1	3	$\times$	0.0	Sigmoid
$D(x)$ – $3 \times 64 \times 64$ input						
Convolution	4×4	2×2	64	$\times$	0.2	Leaky ReLU
Convolution	4×4	1×1	64	$\surd$	0.2	Leaky ReLU
Convolution	4×4	2×2	128	$\surd$	0.2	Leaky ReLU
Convolution	4×4	1×1	128	$\surd$	0.2	Leaky ReLU
Convolution	4×4	2×2	256	$\surd$	0.2	Leaky ReLU
Convolution	4×4	1×1	256	$\surd$	0.2	Leaky ReLU
$D(z)$ – $256 \times 1 \times 1$ input						
Convolution	1×1	1×1	2048	$\times$	0.2	Leaky ReLU
Convolution	1×1	1×1	2048	$\times$	0.2	Leaky ReLU
$D(x, z)$ – $2304 \times 1 \times 1$ input						
Concatenate $D(x)$ and $D(z)$ along the channel axis						
Convolution	1×1	1×1	4096	$\times$	0.2	Leaky ReLU
Convolution	1×1	1×1	4096	$\times$	0.2	Leaky ReLU
Convolution	1×1	1×1	1	$\times$	0.2	Sigmoid
Optimizer	Adam ($\alpha = 10^{-4}$, $\beta_1 = 0.5$, $\beta_2 = 10^{-3}$)					
Batch size	128					
Epochs	125					
Leaky ReLU slope	0.01					
Weight, bias initialization	Isotropic gaussian ($\mu = 0$, $\sigma = 0.01$), Constant(0)					

Table 5.6 – Tiny ImageNet model hyperparameters (unsupervised).

6 Prologue to Third Article

6.1 Article Details

A learned representation for artistic style.

Vincent Dumoulin, Jonathon Shlens, and Manjunath Kudlur. *Proceedings of the International Conference on Learning Representations, 2017.*

Personal Contribution.

I am first contributor to this work with regards to experiment design, analysis and writing.

6.2 Context

At the time this work was published, deep learning-based artistic style transfer was achieved either through an optimization-based procedure on an arbitrary content-style image pair (Gatys et al., 2015b, 2016b), or using a feedforward neural network trained on a single style for an arbitrary content image (Ulyanov et al., 2016a; Li and Wand, 2016; Johnson et al., 2016). Importantly, these approaches did not learn an explicit reprentation for the style they modeled.

6.3 Contributions

In this work we investigate the construction of a single, scalable deep network that can parsimoniously capture the artistic style of a diversity of paintings. We demonstrate that such a network generalizes across a diversity of artistic styles by reducing a painting to a point in an embedding space. Importantly, this model

permits a user to explore new painting styles by arbitrarily combining the styles learned from individual paintings.

6.4 Recent Developments

Follow-up work has explored having the style image inform the conditioning of the style directly rather than through a learned embedding (Ghiasi et al., 2017; Huang and Belongie, 2017).

The conditional instance normalization technique introduced in this work was a direct inspiration for a class of conditioning mechanisms which has found success in speech recognition (Kim et al., 2017), visual question-answering (de Vries et al., 2017), and visual reasoning (Perez et al., 2017a,b).

Artistic style transfer and texture synthesis are still being actively worked on. Recent advances include work on achieving finer-grained control on the stylized image in terms of color, spatial location, and spatial scale of the textures and patterns (Gatys et al., 2017), alternative loss function formulations (Chen and Schmidt, 2016; Li et al., 2017b; Berger and Memisevic, 2017), and GAN-based approaches (Li and Wand, 2016; Jetchev et al., 2016; Bergmann et al., 2017).

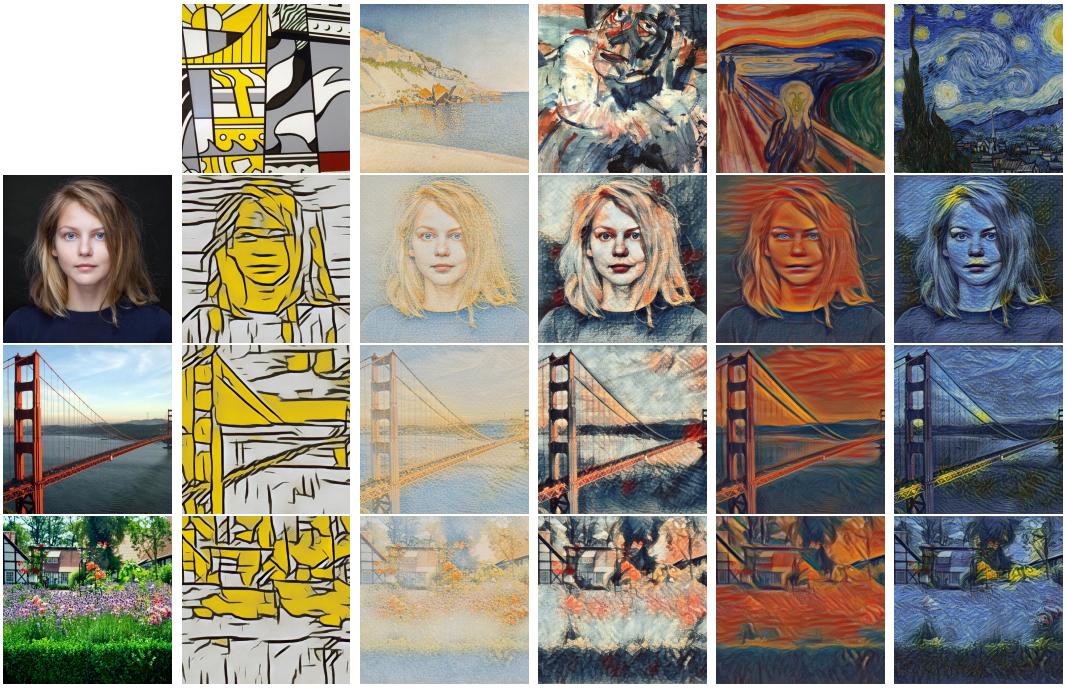
7 A learned representation for artistic style

7.1 Introduction

A pastiche is an artistic work that imitates the style of another one. Computer vision and more recently machine learning have a history of trying to automate pastiche, that is, render an image in the style of another one. This task is called *style transfer*, and is closely related to the texture synthesis task. While the latter tries to capture the statistical relationship between the pixels of a source image which is assumed to have a stationary distribution at some scale, the former does so while also attempting to preserve some notion of content.

On the computer vision side, Efros and Leung (1999) and Wei and Levoy (2000) attempt to "grow" textures one pixel at a time using non-parametric sampling of pixels in an examplar image. Efros and Freeman (2001) and Liang et al. (2001) extend this idea to "growing" textures one patch at a time, and Efros and Freeman (2001) uses the approach to implement "texture transfer", i.e. transfering the texture of an object onto another one. Kwatra et al. (2005) approaches the texture synthesis problem from an energy minimization perspective, progressively refining the texture using an EM-like algorithm. Hertzmann et al. (2001) introduces the concept of "image analogies": given a pair of "unfiltered" and "filtered" versions of an examplar image, a target image is processed to create an analogous "filtered" result. More recently, Frigo et al. (2016) treats style transfer as a local texture transfer (using an adaptive patch partition) followed by a global color transfer, and Elad and Milanfar (2017) extends Kwatra's energy-based method into a style transfer algorithm by taking content similarity into account.

On the machine learning side, it has been shown that a trained classifier can be used as a feature extractor to drive texture synthesis and style transfer. Gatys et al. (2015a) uses the VGG-19 network (Simonyan and Zisserman, 2014) to extract features from a texture image and a synthesized texture. The two sets of features



(a) With conditional instance normalization, a single style transfer network can capture 32 styles at the same time, five of which are shown here. All 32 styles in this single model are in the Appendix. Golden Gate Bridge photograph by Rich Niewiroski Jr.

(b) The style representation learned via conditional instance normalization permits the arbitrary combination of artistic styles. Each pastiche in the sequence corresponds to a different step in interpolating between the γ and β values associated with two styles the model was trained on.

Figure 7.1 – Pastiches produced by a style transfer network trained on 32 styles chosen for their variety.

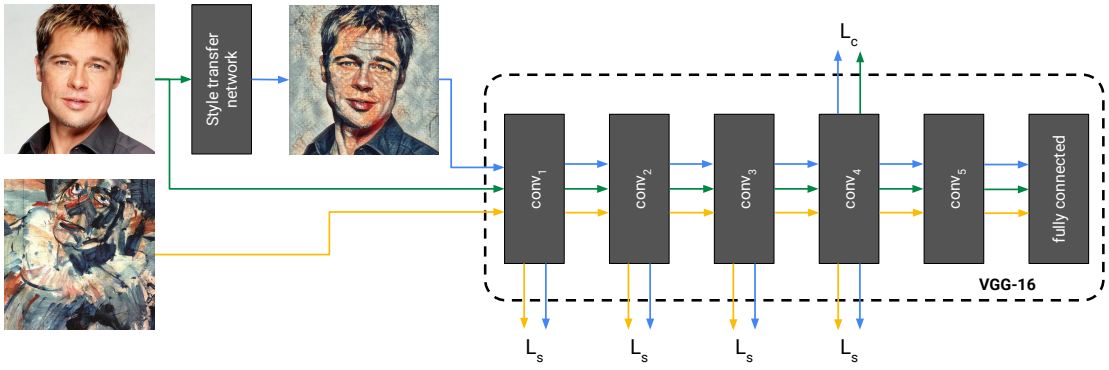


Figure 7.2 – Style transfer network training diagram (Johnson et al., 2016; Ulyanov et al., 2016a). A pastiche image is produced by feeding a content image through the style transfer network. The two images, along with a style image, are passed through a trained classifier, and the resulting intermediate representations are used to compute the content loss $\mathcal{L}_c$ and style loss $\mathcal{L}_s$. The parameters of the classifier are kept fixed throughout training.

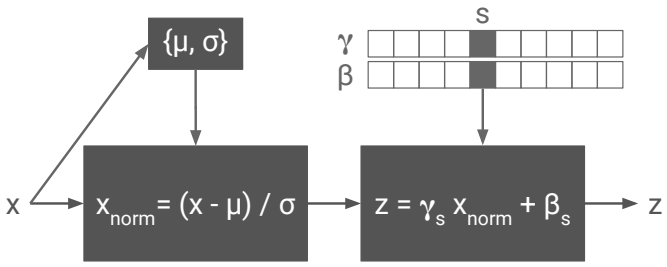


Figure 7.3 – Conditional instance normalization. The input activation x is normalized across both spatial dimensions and subsequently scaled and shifted using style-dependent parameter vectors γ_s, β_s where s indexes the style label.

are compared and the synthesized texture is modified by gradient descent so that the two sets of features are as close as possible. Gatys et al. (2015b, 2016b) extends this idea to style transfer by adding the constraint that the synthesized image also be close to a content image with respect to another set of features extracted by the trained VGG-19 classifier.

While very flexible, this algorithm is expensive to run due to the optimization loop being carried. Ulyanov et al. (2016a), Li and Wand (2016) and Johnson et al. (2016) tackle this problem by introducing a *feedforward style transfer network*, which is trained to go from content to pastiche image in one pass. However, in doing so some of the flexibility of the original algorithm is lost: the style transfer network

is tied to a single style, which means that separate networks have to be trained for every style being modeled. Subsequent work has brought some performance improvements to style transfer networks, e.g. with respect to color preservation (Gatys et al., 2016a) or style transfer quality (Ulyanov et al., 2016b, 2017), but to our knowledge the problem of the single-purpose nature of style transfer networks remains untackled.

We think this is an important problem that, if solved, would have both scientific and practical importance. First, style transfer has already found use in mobile applications, for which on-device processing is contingent upon the models having a reasonable memory footprint. More broadly, building a separate network for each style ignores the fact that individual paintings share many common visual elements and a true model that captures artistic style would be able to exploit and learn from such regularities. Furthermore, the degree to which an artistic styling model might generalize across painting styles would directly measure our ability to build systems that parsimoniously capture the higher level features and statistics of photographs and images (Simoncelli and Olshausen, 2001).

In this work, we show that a simple modification of the style transfer network, namely the introduction of *conditional instance normalization*, allows it to learn multiple styles (Figure 7.1a).We demonstrate that this approach is flexible yet comparable to single-purpose style transfer networks, both qualitatively and in terms of convergence properties. This model reduces each style image into a point in an embedding space. Furthermore, this model provides a generic representation for artistic styles that seems flexible enough to capture new artistic styles much faster than a single-purpose network. Finally, we show that the embeddding space representation permits one to arbitrarily combine artistic styles in novel ways not previously observed (Figure 7.1b).

7.2 Style transfer with deep networks

Style transfer can be defined as finding a pastiche image p whose content is similar to that of a content image c but whose style is similar to that of a style image s. This objective is by nature vaguely defined, because similarity in content and style are themselves vaguely defined.

The neural algorithm of artistic style proposes the following definitions:

— Two images are similar in content if their high-level features as extracted by a trained classifier are close in Euclidian distance.

— Two images are similar in style if their low-level features as extracted by a trained classifier share the same statistics or, more concretely, if the difference between the features' Gram matrices has a small Frobenius norm.

The first point is motivated by the empirical observation that high-level features in classifiers tend to correspond to higher levels of abstractions (see Zeiler and Fergus (2014) for visualizations; see Johnson et al. (2016) for style transfer features). The second point is motivated by the observation that the artistic style of a painting may be interpreted as a visual texture (Gatys et al., 2015a). A visual texture is conjectured to be spatially homogenous and consist of repeated structural motifs whose minimal sufficient statistics are captured by lower order statistical measurements (Julesz, 1962; Portilla and Simoncelli, 2000).

In its original formulation, the neural algorithm of artistic style proceeds as follows: starting from some initialization of p (e.g. c, or some random initialization), the algorithm adapts p to minimize the loss function

$$\mathcal{L}(s, c, p) = \lambda_s \mathcal{L}_s(p) + \lambda_c \mathcal{L}_c(p), \tag{7.1}$$

where $\mathcal{L}_s(p)$ is the style loss, $\mathcal{L}_c(p)$ is the content loss and λ_s, λ_c are scaling hyper-parameters. Given a set of "style layers" $\mathcal{S}$ and a set of "content layers" $\mathcal{C}$, the style and content losses are themselves defined as

$$\mathcal{L}_s(p) = \sum_{i \in \mathcal{S}} \frac{1}{U_i} \mid\mid G(\phi_i(p)) - G(\phi_i(s)) \mid\mid_F^2 \tag{7.2}$$

$$\mathcal{L}_c(p) = \sum_{j \in \mathcal{C}} \frac{1}{U_j} \mid\mid \phi_j(p) - \phi_j(c) \mid\mid_2^2 \tag{7.3}$$

where $\phi_l(x)$ are the classifier activations at layer l, U_l is the total number of units at layer l and $G(\phi_l(x))$ is the Gram matrix associated with the layer l activations. In practice, we set $\lambda_c = 1.0$ and and leave λ_s as a free hyper-parameter.

In order to speed up the procedure outlined above, a feed-forward convolutional network, termed a style transfer network T, is introduced to learn the transformation (Johnson et al., 2016; Li and Wand, 2016; Ulyanov et al., 2016a). It takes as

Figure 7.4 – A single style transfer network was trained to capture the style of 10 Monet paintings, five of which are shown here. All 10 styles in this single model are in the Appendix. Golden Gate Bridge photograph by Rich Niewiroski Jr.

input a content image c and outputs the pastiche image p directly (Figure 7.2). The network is trained on many content images (Deng et al., 2009) using the same loss function as above, i.e.

$$\mathcal{L}(s, c) = \lambda_s \mathcal{L}_s(T(c)) + \lambda_c \mathcal{L}_c(T(c)). \tag{7.4}$$

While feedforward style transfer networks solve the problem of speed at test-time, they also suffer from the fact that the network T is tied to one specific painting style. This means that a separate network T has to be trained for *every* style to be imitated. The real-world impact of this limitation is that it becomes prohibitive to implement a style transfer application on a memory-limited device, such as a smartphone.

7.2.1 N-styles feedforward style transfer networks

Our work stems from the intuition that many styles probably share some degree of computation, and that this sharing is thrown away by training N networks from scratch when building an N-styles style transfer system. For instance, many impressionist paintings share similar paint strokes but differ in the color palette being used. In that case, it seems very wasteful to treat a set of N impressionist paintings as completely separate styles.

To take this into account, we propose to train a single conditional style transfer network $T(c, s)$ for N styles. The conditional network is given both a content image and the identity of the style to apply and produces a pastiche corresponding to that style. While the idea is straightforward on paper, there remains the open question of how conditioning should be done. In exploring this question, we found a very surprising fact about the role of normalization in style transfer networks: to model a style, it is sufficient to specialize scaling and shifting parameters after normalization to each specific style. In other words, all convolutional weights of a style transfer network can be shared across many styles, and it is sufficient to tune parameters for an affine transformation after normalization for each style.

We call this approach *conditional instance normalization*. The goal of the procedure is transform a layer's activations x into a normalized activation z specific to painting style s. Building off the instance normalization technique proposed in Ulyanov et al. (2016b, 2017), we augment the γ and β parameters so that they're $N \times C$ matrices, where N is the number of styles being modeled and C is the number of output feature maps. Conditioning on a style is achieved as follows:

$$z = \gamma_s \left(\frac{x - \mu}{\sigma} \right) + \beta_s \tag{7.5}$$

where μ and σ are x's mean and standard deviation taken across spatial axes and γ_s and β_s are obtained by selecting the row corresponding to s in the γ and β matrices (Figure 7.3). One added benefit of this approach is that one can stylize a single image into N painting styles with a single feed forward pass of the network with a batch size of N. In constrast, a single-style network requires N feed forward passes to perform N style transfers (Johnson et al., 2016; Li and Wand, 2016; Ulyanov et al., 2016a).

Because conditional instance normalization only acts on the scaling and shifting

parameters, training a style transfer network on N styles requires fewer parameters than the naive approach of training N separate networks. In a typical network setup, the model consists of roughly 1.6M parameters, only around 3K (or 0.2%) of which specify individual artistic styles. In fact, because the size of γ and β grows linearly with respect to the number of feature maps in the network, this approach requires $O(N \times L)$ parameters, where L is the total number of feature maps in the network.

In addition, as is discussed in subsection 7.3.4, conditional instance normalization presents the advantage that integrating an $N + 1^{th}$ style to the network is cheap because of the very small number of parameters to train.

7.3 Experimental results

7.3.1 Methodology

Unless noted otherwise, all style transfer networks were trained using the hyperparameters outlined in the Appendix's Table 7.1.

We used the same network architecture as in Johnson et al. (2016), except for two key details: zero-padding is replaced with mirror-padding, and transposed convolutions (also sometimes called *deconvolutions*) are replaced with nearest-neighbor upsampling followed by a convolution. The use of mirror-padding avoids border patterns sometimes caused by zero-padding in SAME-padded convolutions, while the replacement for transposed convolutions avoids checkerboard patterning, as discussed in in Odena et al. (2016). We find that with these two improvements training the network no longer requires a total variation loss that was previously employed to remove high frequency noise as proposed in Johnson et al. (2016).

Our training procedure follows Johnson et al. (2016). Briefly, we employ the ImageNet dataset (Deng et al., 2009) as a corpus of training content images. We train the N-style network with stochastic gradient descent using the Adam optimizer (Kingma and Ba, 2014). Details of the model architecture are in the Appendix. A complete implementation of the model in TensorFlow (Abadi et al., 2015) as well as a pretrained model are available for download [i]. The evaluation images used

i. `https://github.com/tensorflow/magenta`

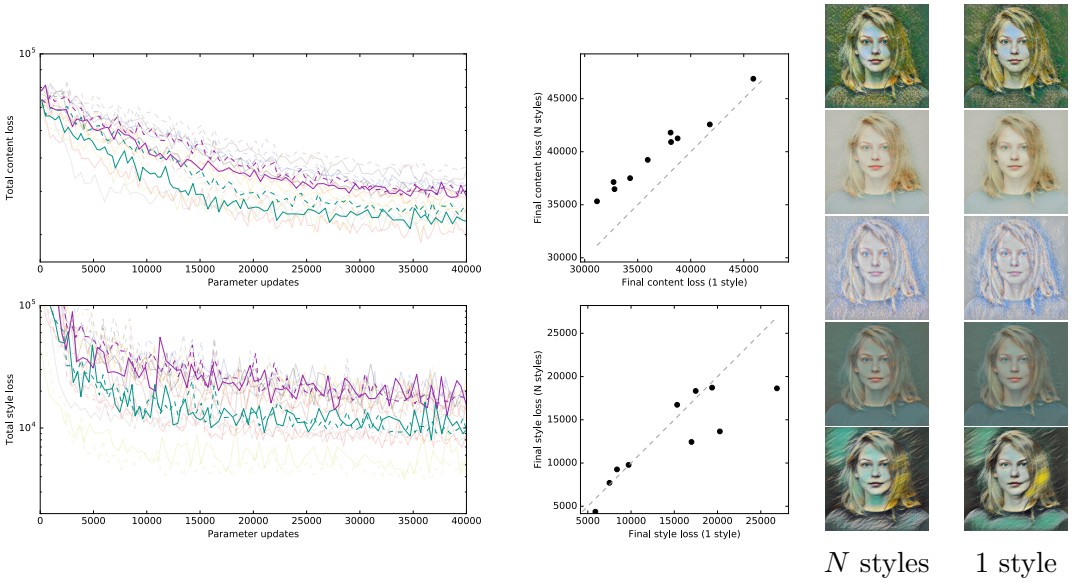


Figure 7.5 – The N-styles model exhibits learning dynamics comparable to individual models. (Left column) The N-styles model converges slightly slower in terms of content loss (top) and as fast in terms of style loss (bottom) than individual models. Training on a single Monet painting is represented by two curves with the same color. The dashed curve represents the N-styles model, and the full curves represent individual models. Emphasis has been added on the styles for *Vetheuil (1902)* (teal) and *Water Lilies* (purple) for visualization purposes; remaining colors correspond to other Monet paintings (see Appendix). (Center column) The N-styles model reaches a slightly higher final content loss than (top, $8.7 \pm 3.9\%$ increase) and a final style loss comparable to (bottom, $8.9 \pm 16.5\%$ decrease) individual models. (Right column) Pastiches produced by the N-styles network are qualitatively comparable to those produced by individual networks.

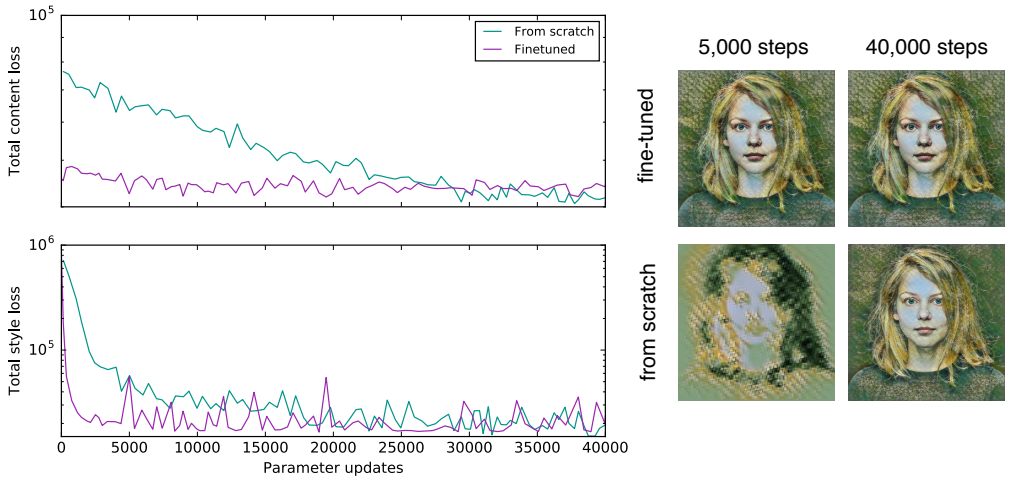


Figure 7.6 – The trained network is efficient at learning new styles. (Left column) Learning γ and β from a trained style transfer network converges much faster than training a model from scratch. (Right) Learning γ and β for 5,000 steps from a trained style transfer network produces pastiches comparable to that of a single network trained from scratch for 40,000 steps. Conversely, 5,000 step of training from scratch produces leads to a poor pastiche.

for this work were resized such that their smaller side has size 512. Their stylized versions were then center-cropped to 512x512 pixels for display.

7.3.2 Training a single network on N styles produces stylizations comparable to independently-trained models

As a first test, we trained a 10-styles model on stylistically similar images, namely 10 impressionist paintings from Claude Monet. Figure 7.4 shows the result of applying the trained network on evaluation images for a subset of the styles, with the full results being displayed in the Appendix. The model captures different color palettes and textures. We emphasize that 99.8% of the parameters are shared across all styles in contrast to 0.2% of the parameters which are unique to each painting style.

To get a sense of what is being traded off by folding 10 styles into a single network, we trained a separate, single-style network on each style and compared them to the 10-styles network in terms of style transfer quality and training speed (Figure 7.5).

The left column compares the learning curves for style and content losses be-

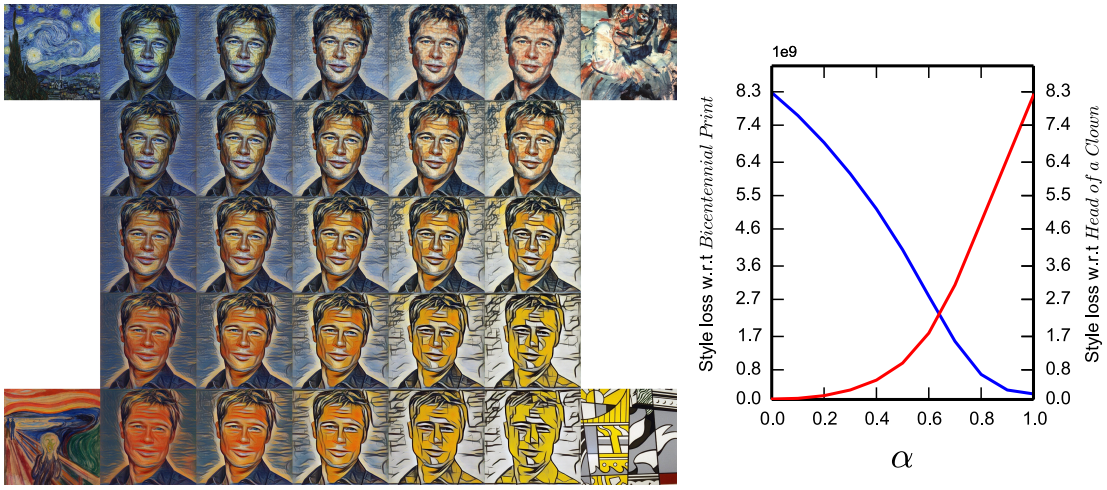


Figure 7.7 – The N-styles network can arbitrarily combine artistic styles. (Left) Combining four styles, shown in the corners. Each pastiche corresponds to a different convex combination of the four styles' γ and β values. (Right) As we transition from one style to another (*Bicentennial Print* and *Head of a Clown* in this case), the style losses vary monotonically.

tween the single-style networks and the 10-styles network. The losses were averaged over 32 random batches of content images. By visual inspection, we observe that the 10-styles network converges as quickly as the single-style networks in terms of style loss, but lags slightly behind in terms of content loss.

In order to quantify this observation, we compare the final losses for 10-styles and single-style models (center column). The 10-styles network's content loss is around $8.7 \pm 3.9\%$ higher than its single-style counterparts, while the difference in style losses ($8.9 \pm 16.5\%$ lower) is insignificant. While the N-styles network suffers from a slight decrease in content loss convergence speed, this may not be a fair comparison, given that it takes N times more parameter updates to train N single-style networks separately than to train them with an N-styles network.

The right column shows a comparison between the pastiches produced by the 10-styles network and the ones produced by the single-style networks. We see that both results are qualitatively similar.

7.3.3 The N-styles model is flexible enough to capture very different styles

We evaluated the flexibility of the N-styles model by training a style transfer network on 32 works of art chosen for their diversity. Figure 7.1a shows the result of applying the trained network on evaluation images for a subset of the styles. Once again, the full results are displayed in the Appendix. The model appears to be capable of modeling all 32 styles in spite of the tremendous variation in color palette and the spatial scale of the painting styles.

7.3.4 The trained network generalizes across painting styles

Since all weights in the transformer network are shared between styles, one way to incorporate a new style to a trained network is to keep the trained weights fixed and learn a new set of γ and β parameters. To test the efficiency of this approach, we used it to incrementally incorporate Monet's *Plum Trees in Blossom* painting to the network trained on 32 varied styles. Figure 7.6 shows that doing so is much faster than training a new network from scratch (left) while yielding comparable pastiches: even after eight times fewer parameter updates than its single-style counterpart, the fine-tuned model produces comparable pastiches (right).

7.3.5 The trained network can arbitrarily combine painting styles

The conditional instance normalization approach raises some interesting questions about style representation. In learning a different set of γ and β parameters for every style, we are in some sense learning an embedding of styles.

Previous work suggested that cleverly balancing optimization strategies offers an opportunity to blend painting styles [i]. To probe the utility of this embedding, we tried convex combinations of the γ and β values to blend very distinct painting styles (Figure 7.1b; Figure 7.7, left column). Employing a single convex combination produces a smooth transition from one style to the other. Suppose (γ_1, β_1) and (γ_2, β_2) are the parameters corresponding to two different styles. We use $\gamma = \alpha \times \gamma_1 + (1 - \alpha) \times \gamma_2$ and $\beta = \alpha \times \beta_1 + (1 - \alpha) \times \beta_2$ to stylize an image.

i. For instance, `https://github.com/jcjohnson/neural-style`

Employing convex combinations may be extended to an arbitrary number of styles [i]. Figure 7.7 (right column) shows the style loss from the transformer network for a given source image, with respect to the *Bicentennial Print* and *Head of a Clown* paintings, as we vary α from 0 to 1. As α increases, the style loss with respect to *Bicentennial Print* increases, which explains the smooth fading out of that style's artifact in the transformed image.

7.4 Discussion

It seems surprising that such a small proportion of the network's parameters can have such an impact on the overall process of style transfer. A similar intuition has been observed in auto-regressive models of images (van den Oord et al., 2016c) and audio (van den Oord et al., 2016a) where the conditioning process is mediated by adjusting the biases for subsequent samples from the model. That said, in the case of art stylization when posed as a feedforward network, it could be that the specific network architecture is unable to take full advantage of its capacity. We see evidence for this behavior in that pruning the architecture leads to qualitatively similar results. Another interpretation could be that the convolutional weights of the style transfer network encode transformations that represent "elements of style". The scaling and shifting factors would then provide a way for each style to inhibit or enhance the expression of various elements of style to form a global identity of style. While this work does not attempt to verify this hypothesis, we think that this would constitute a very promising direction of research in understanding the computation behind style transfer networks as well as the representation of images in general.

Concurrent to this work, Gatys et al. (2017) demonstrated exciting new methods for revising the loss to selectively adjust the spatial scale, color information and spatial localization of the artistic style information. These methods are complementary to the results in this paper and present an interesting direction for exploring how spatial and color information uniquely factor into artistic style representation.

i. Please see the code repository for real-time, interactive demonstration. A screen capture is available at `https://www.youtube.com/watch?v=6ZHiARZmiUI`.

The question of how predictive each style image is of its corresponding style representation is also of great interest. If it is the case that the style representation can easily be predicted from a style image, one could imagine building a transformer network which skips learning an individual conditional embedding and instead learn to produce a pastiche directly from a style and a content image, much like in the original neural algorithm of artistic style, but without any optimization loop at test time.

Finally, the learned style representation opens the door to generative models of style: by modeling enough paintings of a given artistic movement (e.g. impressionism), one could build a collection of style embeddings upon which a generative model could be trained. At test time, a style representation would be sampled from the generative model and used in conjunction with the style transfer network to produce a random pastiche of that artistic movement.

In summary, we demonstrated that conditional instance normalization constitutes a simple, efficient and scalable modification of style transfer networks that allows them to model multiple styles at the same time. A practical consequence of this approach is that a new painting style may be transmitted to and stored on a mobile device with a small number of parameters. We showed that despite its simplicity, the method is flexible enough to capture very different styles while having very little impact on training time and final performance of the trained network. Finally, we showed that the learned representation of style is useful in arbitrarily combining artistic styles. This work suggests the existence of a learned representation for artistic styles whose vocabulary is flexible enough to capture a diversity of the painted world.

Acknowledgments

We would like to thank Fred Bertsch, Douglas Eck, Cinjon Resnick and the rest of the Google Magenta team for their feedback; Peyman Milanfar, Michael Elad, Feng Yang, Jon Barron, Bhavik Singh, Jennifer Daniel as well as the the Google Brain team for their crucial suggestions and advice; an anonymous reviewer for helpful suggestions about applying this model in a mobile domain. Finally, we would like to thank the Google Cultural Institute, whose curated collection of art photographs was very helpful in finding exciting style images to train on.

Appendix

Operation	Kernel size	Stride	Channels	Padding	Nonlinearity
Network – $256 \times 256 \times 3$ input					
Convolution	9	1	32	SAME	ReLU
Convolution	3	2	64	SAME	ReLU
Convolution	3	2	128	SAME	ReLU
Res. block			128		
Res. block			128		
Res. block			128		
Res. block			128		
Res. block			128		
Upsampling			64		
Upsampling			32		
Convolution	9	1	3	SAME	Sigmoid
Res. block – C channels					
Convolution	3	1	C	SAME	ReLU
Convolution	3	1	C	SAME	Linear
	Add the input and the output				
Upsampling – C channels					
	Nearest-neighbor interpolation, factor 2				
Convolution	3	1	C	SAME	ReLU

Padding mode	REFLECT
Normalization	Conditional instance normalization after every convolution
Optimizer	Adam (Kingma and Ba, 2014)
Optimizer hyperparameters	$\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$
Parameter updates	40,000
Batch size	16
Weight initialization	Isotropic gaussian ($\mu = 0$, $\sigma = 0.01$)

Table 7.1 – Style transfer network hyperparameters.

Claude Monet, *Grainstacks at Giverny; the Evening Sun* (1888/1889).

Claude Monet, *Plum Trees in Blossom* (1879).

Claude Monet, *Poppy Field* (1873).

Claude Monet, *Rouen Cathedral, West Façade* (1894).

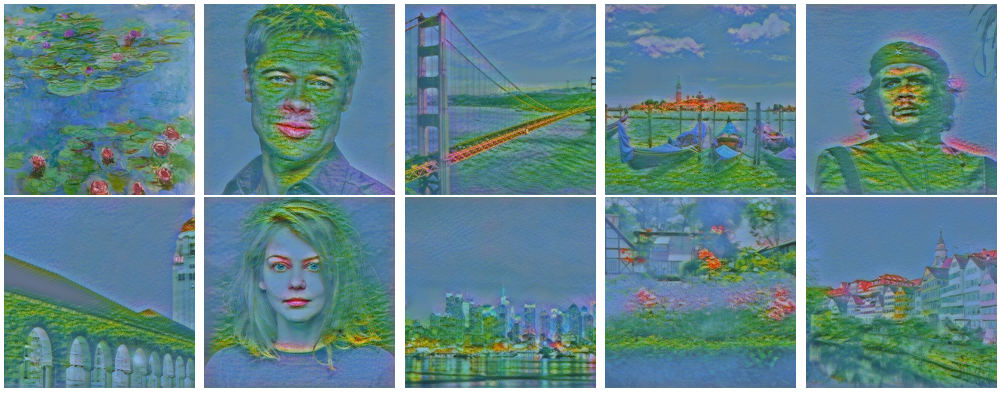
Claude Monet, *Sunrise (Marine)* (1873).

Claude Monet, *The Road to Vétheuil* (1879).

Claude Monet, *Three Fishing Boats* (1886).

Claude Monet, *Vétheuil* (1879).

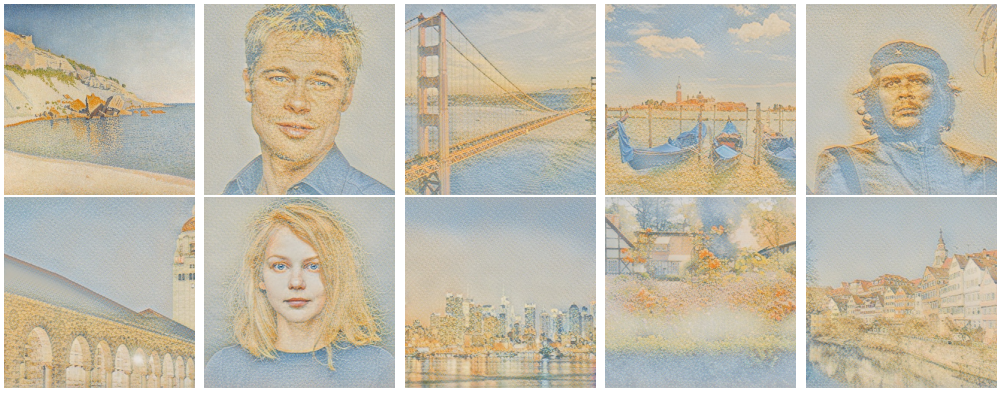
Claude Monet, *Vétheuil* (1902).



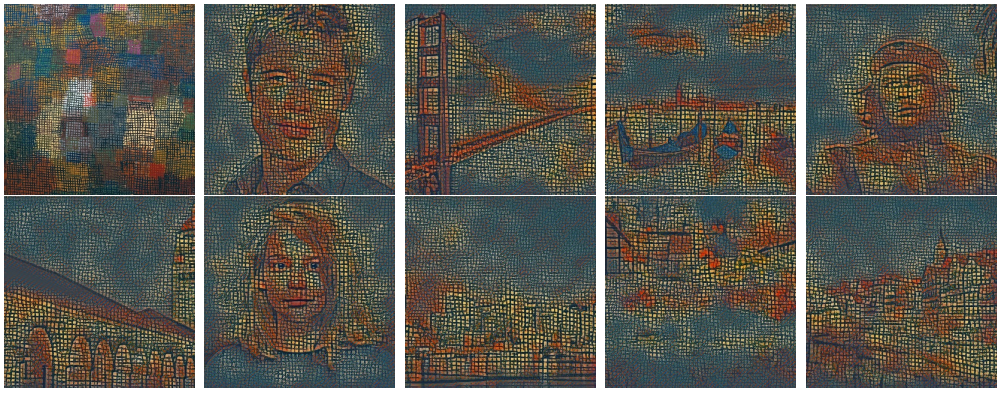
Claude Monet, *Water Lilies* (ca. 1914-1917).

Roy Lichtenstein, *Bicentennial Print* (1975).

Ernst Ludwig Kirchner, *Boy with Sweets* (1918).



Paul Signac, *Cassis, Cap Lombard, Opus 196* (1889).



Paul Klee, *Colors from a Distance* (1932).

Frederic Edwin Church, *Cotopaxi* (1855).

Jamini Roy, *Crucifixion.*

Henri de Toulouse-Lautrec, *Divan Japonais* (1893).

Egon Schiele, *Edith with Striped Dress, Sitting* (1915).

Georges Rouault, *Head of a Clown* (ca. 1907-1908).

William Hoare, *Henry Hoare, "The Magnificent", of Stourhead* (about 1750-1760).

Giorgio de Chirico, *Horses on the seashore* (1927/1928).

Vincent van Gogh, *Landscape at Saint-Rémy (Enclosed Field with Peasant)* (1889).

Nicolas Poussin, *Landscape with a Calm* (1650-1651).

Bernardino Fungai, *Madonna and Child with Two Hermit Saints* (early 1480s).

Max Hermann Maxy, *Portrait of a Friend* (1926).

Juan Gris, *Portrait of Pablo Picasso* (1912).

Severini Gino, *Ritmo plastico del 14 luglio* (1913).

Richard Diebenkorn, *Seawall* (1957).

Alice Bailly, *Self-Portrait* (1917).

Grayson Perry, *The Annunciation of the Virgin Deal* (2012).

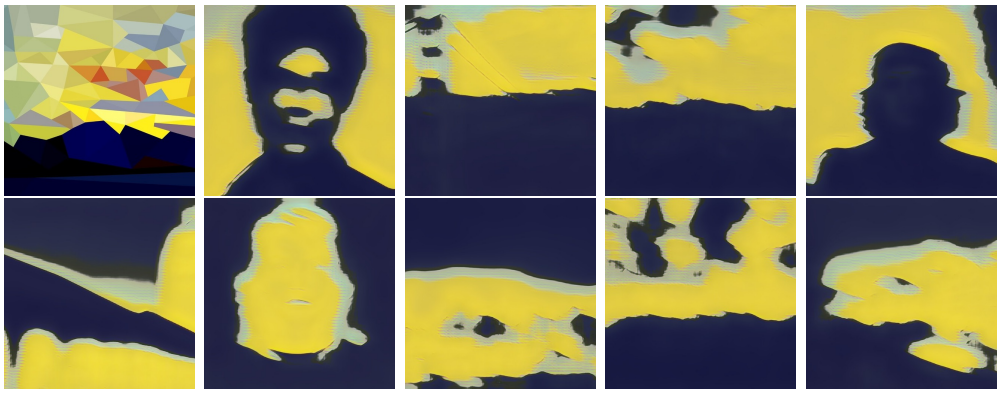
William Glackens, *The Green Boathouse* (ca. 1922).

Edvard Munch, *The Scream* (1910).

Vincent van Gogh, *The Starry Night* (1889).

Pieter Bruegel the Elder, *The Tower of Babel* (1563).

Wolfgang Lettl, *The Trial* (1981).



Douglas Coupland, *Thomson No. 5 (Yellow Sunset)* (2011).

Claude Monet, *Three Fishing Boats* (1886).

John Ruskin, *Trees in a Lane* (1847).

Giuseppe Cades, *Tullia about to Ride over the Body of Her Father in Her Chariot* (about 1770-1775).

Berthe Morisot, *Under the Orange Tree* (1889).

Giulio Romano (Giulio Pippi), *Victory, Janus, Chronos and Gaea* (about 1532-1534).

Wassily Kandinsky, *White Zig Zags* (1922).

8 Discussion

The articles presented in this thesis are inspired by and seek to advance knowledge in the field of representation learning in one way or another.

The simulation work on physical implementations of restricted Boltzmann machines gives insight into the way in which RBM performance degrades when subjected to real-world constraints, such as noise on parameters, limited parameter amplitude and sparse connectivity constraints. While research efforts have shifted from energy-based models towards directed probabilistic graphical models over the past few years, dedicated hardware is exactly the kind of innovation which may spark a renewed interest in those models, and the conclusions presented in this work are likely to be relevant to any effort in that direction.

The work on adversarial inference in GANs provides access to the representation learned by the generator, allowing its re-use for auxiliary tasks such as semi-supervised learning and image manipulation in an abstract space. While this work does not specifically address other current open GAN problems, such as consistent convergence to an equilibrium point of the value function, mode collapse, or finding an objective measure of performance, it does provide an inference framework which will remain usable even when those problems are resolved.

Finally, the work on artistic style transfer focuses on learning a representation of artistic style, which opens the door to new exciting directions of research. Analysis on the learned representation of a large collection of paintings — e.g., through clustering or low-dimensional projections — may reveal interesting connections between works of art in a purely automated fashion. Manipulating and comparing the learned representations between different artistic styles may help explain the computation carried out by intermediary layers in the style transfer network. This work also had an impact outside of the field of artistic style transfer, with the realization that the conditioning mechanism it introduces can be applied to many different problem settings. In fact, a very general view emerges in the form of a learned representation of *tasks* — in the case of artistic style transfer, a task corresponds

to imitating a particular style — where the representation itself is indicative of the computation required to solve a given task. This shift in perspective may provide valuable insight on how different tasks relate to each other, and help rationalize the computational behaviour of the conditioned networks.

Bibliography

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems.

Bastien, F., Lamblin, P., Pascanu, R., Bergstra, J., Goodfellow, I., Bergeron, A., Bouchard, N., Warde-Farley, D., and Bengio, Y. (2012). Theano: New features and speed improvements. In *Proceedings of the NIPS Workshop on Deep Learning and Unsupervised Feature Learning*.

Belghazi, I., Baratin, A., Rajeswar, S., Ozair, S., Bengio, Y., Courville, A., and Hjelm, D. (2018a). Mutual information neural estimation. In *Proceedings of the International Conference on Machine Learning*.

Belghazi, M. I., Rajeswar, S., Mastropietro, O., Rostamzadeh, N., Mitrovic, J., and Courville, A. (2018b). Hierarchical adversarially learned inference. arXiv:1802.01071.

Bengio, Y. (2012). Deep learning of representations for unsupervised and transfer learning. In *Proceedings of the ICML Workshop on Unsupervised and Transfer Learning*.

Bengio, Y., Léonard, N., and Courville, A. (2013). Estimating or propagating gradients through stochastic neurons for conditional computation. arXiv:1308.3432.

Bengio, Y., Thibodeau-Laufer, E., Alain, G., and Yosinski, J. (2014). Deep gener-

ative stochastic networks trainable by backprop. In *Proceedings of the International Conference on Machine Learning*.

Berger, G. and Memisevic, R. (2017). Incorporating long-range consistency in CNN-based texture generation. In *Proceedings of the International Conference on Learning Representations*.

Bergmann, U., Jetchev, N., and Vollgraf, R. (2017). Learning texture manifolds with the periodic spatial GAN. In *Proceedings of the International Conference on Machine Learning*.

Bergstra, J., Breuleux, O., Bastien, F., Lamblin, P., Pascanu, R., Desjardins, G., Turian, J., Warde-Farley, D., and Bengio, Y. (2010). Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python in Science Conference*.

Bousquet, O. and Bottou, L. (2008). The tradeoffs of large scale learning. In *Advances in Neural Information Processing Systems*.

Brock, A., Lim, T., Ritchie, J., and Weston, N. (2017). Neural photo editing with introspective adversarial networks. In *Proceedings of the International Conference on Learning Representations*.

Chen, T. Q. and Schmidt, M. (2016). Fast patch-based style transfer of arbitrary style. arXiv:1612.04337.

Chen, X., Duan, Y., Houthooft, R., Schulman, J., Sutskever, I., and Abbeel, P. (2016). InfoGAN: Interpretable representation learning by information maximizing generative adversarial nets. In *Advances in Neural Information Processing Systems*.

Cho, K., Raiko, T., and Ilin, A. (2010). Parallel tempering is efficient for learning restricted boltzmann machines. In *Proceedings of the International Joint Conference on Neural Networks*.

Coates, A. and Ng, A. Y. (2011). The importance of encoding versus training with sparse coding and vector quantization. In *Proceedings of the International Conference on Machine Learning*.

Creswell, A., White, T., Dumoulin, V., Arulkumaran, K., Sengupta, B., and Bharath, A. A. (2018). Generative adversarial networks: An overview. *IEEE Signal Processing Magazine*, 35(1):53–65.

de Vries, H., Strub, F., Mary, J., Larochelle, H., Pietquin, O., and Courville, A. (2017). Modulating early visual processing by language. In *Advances in Neural Information Processing Systems*.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). ImageNet: A large-scale hierarchical image database. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*.

Denil, M. and De Freitas, N. (2011). Toward the implementation of a quantum RBM. In *Proceedings of the NIPS Workshop on Deep Learning and Unsupervised Feature Learning*.

Desjardins, G., Courville, A. C., Bengio, Y., Vincent, P., and Delalleau, O. (2010). Tempered Markov chain Monte Carlo for training of restricted Boltzmann machines. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*.

Donahue, J., Krähenbühl, P., and Darrell, T. (2017). Adversarial feature learning. In *Proceedings of the International Conference on Learning Representations*.

Dosovitskiy, A. and Brox, T. (2016). Generating images with perceptual similarity metrics based on deep networks. In *Advances in Neural Information Processing Systems*.

Dumoulin, V. (2014). Morphing faces. https://vdumoulin.github.io/morphing_faces.

Dumoulin, V., Belghazi, I., Poole, B., Lamb, A., Arjovsky, M., Mastropietro, O., and Courville, A. (2017a). Adversarially learned inference. In *Proceedings of the International Conference on Learning Representations*.

Dumoulin, V., Goodfellow, I. J., Courville, A., and Bengio, Y. (2014). On the challenges of physical implementations of RBMs. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

Dumoulin, V., Shlens, J., and Kudlur, M. (2017b). A learned representation for artistic style. In *Proceedings of the International Conference on Learning Representations*.

Dumoulin, V. and Visin, F. (2016). A guide to convolution arithmetic for deep learning. arXiv:1603.07285.

Dupret, A., Belhaire, E., Rodier, J.-C., Lalanne, P., Prévost, D., Garda, P., and Chavel, P. (1996). An optoelectronic CMOS circuit implementing a simulated annealing algorithm. *IEEE Journal of Solid-State Circuits*, 31.

Efros, A. A. and Freeman, W. T. (2001). Image quilting for texture synthesis and transfer. In *Proceedings of the Conference on Computer Graphics and Interactive Techniques*.

Efros, A. A. and Leung, T. K. (1999). Texture synthesis by non-parametric sampling. In *Proceedings of the IEEE Conference on Computer Vision*.

Elad, M. and Milanfar, P. (2017). Style-transfer via texture-synthesis. *IEEE Transactions on Image Processing*, 26.

Fedus, W., Rosca, M., Lakshminarayanan, B., Dai, A. M., Mohamed, S., and Goodfellow, I. (2018). Many paths to equilibrium: GANs do not need to decrease a divergence at every step. In *Proceedings of the International Conference on Learning Representations*.

Fisher, R. A. (1936). The use of multiple measurements in taxonomic problems. *Annals of eugenics*, 7.

Frigo, O., Sabater, N., Delon, J., and Hellier, P. (2016). Split and match: Example-based adaptive patch sampling for unsupervised style transfer. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

Gatys, L., Ecker, A. S., and Bethge, M. (2015a). Texture synthesis using convolutional neural networks. In *Advances in Neural Information Processing Systems*.

Gatys, L. A., Bethge, M., Hertzmann, A., and Shechtman, E. (2016a). Preserving color in neural artistic style transfer. arXiv:1606.05897.

Gatys, L. A., Ecker, A. S., and Bethge, M. (2015b). A neural algorithm of artistic style. arXiv:1508.06576.

Gatys, L. A., Ecker, A. S., and Bethge, M. (2016b). Image style transfer using convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2414–2423.

Gatys, L. A., Ecker, A. S., Bethge, M., Hertzmann, A., and Shechtman, E. (2017). Controlling perceptual factors in neural style transfer. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*.

Germain, M., Gregor, K., Murray, I., and Larochelle, H. (2015). MADE: Masked autoencoder for distribution estimation. In *Proceedings of the International Conference on Machine Learning*.

Ghiasi, G., Lee, H., Kudlur, M., Dumoulin, V., and Shlens, J. (2017). Exploring the structure of a real-time, arbitrary neural artistic stylization network. In *Proceedings of the British Machine Vision Conference*.

Goodfellow, I. (2016). NIPS 2016 tutorial: Generative adversarial networks. Presented at the Neural Information Processing Systems Conference.

Goodfellow, I., Mirza, M., Courville, A., and Bengio, Y. (2013a). Multi-prediction deep Boltzmann machines. In *Advances in Neural Information Processing Systems*.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. In *Advances in Neural Information Processing Systems*.

Goodfellow, I. J., Warde-Farley, D., Mirza, M., Courville, A., and Bengio, Y. (2013b). Maxout networks. In *Proceedings of the International Conference on Machine Learning*.

Griewank, A. and Walther, A. (2008). *Evaluating derivatives: Principles and techniques of algorithmic differentiation*, volume 105. Siam.

Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., and Courville, A. (2017). Improved training of Wasserstein GANs. In *Advances in Neural Information Processing Systems*.

Hertzmann, A., Jacobs, C. E., Oliver, N., Curless, B., and Salesin, D. H. (2001). Image analogies. In *Proceedings of the Conference on Computer Graphics and Interactive Techniques*.

Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., Klambauer, G., and Hochreiter, S. (2017). GANs trained by a two time-scale update rule converge to a Nash equilibrium. In *Advances in Neural Information Processing Systems*.

Hinton, G. E. (2002). Training products of experts by minimizing contrastive divergence. *Neural computation*, 14.

Hinton, G. E., Osindero, S., and Teh, Y.-W. (2006). A fast learning algorithm for deep belief nets. *Neural computation*, 18.

Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. arXiv:1207.0580.

Huang, X. and Belongie, S. (2017). Arbitrary style transfer in real-time with adaptive instance normalization. In *Proceedings of the International Conference on Computer Vision*.

Huszár, F. (2017). Variational inference using implicit distributions. arXiv:1702.08235.

Ising, E. (1925). Beitrag zur theorie des ferromagnetismus. *Zeitschrift für Physik A Hadrons and Nuclei*, 31.

Jetchev, N., Bergmann, U., and Vollgraf, R. (2016). Texture synthesis with spatial generative adversarial networks. In *Proceedings of the NIPS Workshop on Adversarial Training*.

Johnson, J., Alahi, A., and Fei-Fei, L. (2016). Perceptual losses for real-time style transfer and super-resolution. In *Proceedings of the European Conference on Computer Vision*.

Julesz, B. (1962). Visual pattern discrimination. *IRE transactions on Information Theory*, 8.

Karaletsos, T. (2016). Adversarial message passing for graphical models. In *Advances in Neural Information Processing Systems*.

Kim, T., Song, I., and Bengio, Y. (2017). Dynamic layer normalization for adaptive neural acoustic modeling in speech recognition. In *Interspeech*.

Kingma, D. and Ba, J. (2014). Adam: A method for stochastic optimization. In *Proceedings of the International Conference on Learning Representations*.

Kingma, D. P. (2013). Fast gradient-based inference with continuous latent variable models in auxiliary form. arXiv:1306.0733.

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*.

Kingma, D. P., Salimans, T., and Welling, M. (2016). Improving variational inference with inverse autoregressive flow. In *Advances in Neural Information Processing Systems*.

Kingma, D. P. and Welling, M. (2014). Auto-encoding variational bayes. In *Proceedings of the International Conference on Learning Representations*.

Kodali, N., Abernethy, J., Hays, J., and Kira, Z. (2017). On convergence and stability of GANs. arXiv:1705.07215.

Krizhevsky, A. and Hinton, G. (2009). Learning multiple layers of features from tiny images. Technical report, University of Toronto.

Kwatra, V., Essa, I., Bobick, A., and Kwatra, N. (2005). Texture optimization for example-based synthesis. *ACM Transactions on Graphics (ToG)*, 24.

Lamb, A., Dumoulin, V., and Courville, A. (2016). Discriminative regularization for generative models. arXiv:1602.03220.

Lamb, A. M., Hjelm, D., Ganin, Y., Cohen, J. P., Courville, A. C., and Bengio, Y. (2017). GibbsNet: Iterative adversarial inference for deep graphical models. In *Advances in Neural Information Processing Systems*.

Larochelle, H., Bengio, Y., and Turian, J. (2010). Tractable multivariate binary density estimation and the restricted Boltzmann forest. *Neural computation*, 22.

Larochelle, H. and Murray, I. (2011). The neural autoregressive distribution estimator. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*.

Larsen, A. B. L., Sønderby, S. K., and Winther, O. (2016). Autoencoding beyond pixels using a learned similarity metric. In *Proceedings of the International Conference on Machine Learning*.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86.

Li, C., Liu, H., Chen, C., Pu, Y., Chen, L., Henao, R., and Carin, L. (2017a). Alice: Towards understanding adversarial learning for joint distribution matching. In *Advances in Neural Information Processing Systems*.

Li, C. and Wand, M. (2016). Precomputed real-time texture synthesis with markovian generative adversarial networks. In *Proceedings of the European Conference on Computer Vision*.

Li, Y., Fang, C., Yang, J., Wang, Z., Lu, X., and Yang, M.-H. (2017b). Universal style transfer via feature transforms. In *Advances in Neural Information Processing Systems*.

Liang, L., Liu, C., Xu, Y.-Q., Guo, B., and Shum, H.-Y. (2001). Real-time texture synthesis by patch-based sampling. *ACM Transactions on Graphics*, 20.

Lin, J. (1991). Divergence measures based on the Shannon entropy. *IEEE Transactions on Information Theory*, 37.

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*.

Long, P. M. and Servedio, R. (2010). Restricted Boltzmann machines are hard to approximately evaluate or simulate. In *Proceedings of the International Conference on Machine Learning*.

Lucic, M., Kurach, K., Michalski, M., Gelly, S., and Bousquet, O. (2017). Are GANs created equal? A large-scale study. arXiv:1711.10337.

Maaløe, L., Sønderby, C. K., Sønderby, S. K., and Winther, O. (2016). Auxiliary deep generative models. In *Proceedings of the International Conference on Machine Learning*.

Makhzani, A. (2018). Implicit autoencoders. arXiv:1805.09804.

Makhzani, A., Shlens, J., Jaitly, N., and Goodfellow, I. (2015). Adversarial autoencoders. arXiv:1511.05644.

Mescheder, L., Nowozin, S., and Geiger, A. (2017). Adversarial variational Bayes: Unifying variational autoencoders and generative adversarial networks. In *Proceedings of the International Conference on Machine Learning*.

Nagarajan, V. and Kolter, J. Z. (2017). Gradient descent GAN optimization is locally stable. In *Advances in Neural Information Processing Systems*.

Netzer, Y., Wang, T., Coates, A., Bissacco, A., Wu, B., and Ng, A. Y. (2011). Reading digits in natural images with unsupervised feature learning. In *Proceedings of the NIPS Workshop on Deep Learning and Unsupervised Feature Learning*.

Odena, A., Dumoulin, V., and Olah, C. (2016). Deconvolution and checkerboard artifacts. *Distill*.

Perez, E., de Vries, H., Strub, F., Dumoulin, V., and Courville, A. (2017a). Learning visual reasoning without strong priors. In *Proceedings of the ICML Workshop on Machine Learning in Speech and Language Processing*.

Perez, E., Strub, F., de Vries, H., Dumoulin, V., and Courville, A. (2017b). FiLM: Visual reasoning with a general conditioning layer. arXiv:1709.07871.

Portilla, J. and Simoncelli, E. P. (2000). A parametric texture model based on joint statistics of complex wavelet coefficients. *International Journal of Computer Vision*, 40.

Radford, A., Metz, L., and Chintala, S. (2016). Unsupervised representation learning with deep convolutional generative adversarial networks. In *Proceedings of the International Conference on Learning Representations*.

Rasmus, A., Valpola, H., Honkala, M., Berglund, M., and Raiko, T. (2015). Semi-supervised learning with ladder network. In *Advances in Neural Information Processing Systems*.

Rezende, D. J., Mohamed, S., and Wierstra, D. (2014). Stochastic backpropagation and approximate inference in deep generative models. In *Proceedings of the International Conference on Machine Learning*.

Roth, K., Lucchi, A., Nowozin, S., and Hofmann, T. (2017). Stabilizing training of generative adversarial networks through regularization. In *Advances in Neural Information Processing Systems*.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1988). Learning representations by back-propagating errors. *Cognitive modeling*, 5.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., et al. (2015). ImageNet large scale visual recognition challenge. *International Journal of Computer Vision*, 115.

Salakhutdinov, R. (2010). Learning deep Boltzmann machines using adaptive MCMC. In *Proceedings of the International Conference on Machine Learning*.

Salakhutdinov, R. and Murray, I. (2008). On the quantitative analysis of deep belief networks. In *Proceedings of the International Conference on Machine Learning*.

Salakhutdinov, R. R. (2009). Learning in Markov random fields using tempered transitions. In *Advances in Neural Information Processing Systems*.

Salimans, T., Goodfellow, I. J., Zaremba, W., Cheung, V., Radford, A., and Chen, X. (2016). Improved techniques for training GANs. In *Advances in Neural Information Processing Systems*.

Shi, W., Caballero, J., Theis, L., Huszar, F., Aitken, A., Ledig, C., and Wang, Z. (2016). Is the deconvolution layer the same as a convolutional layer? arXiv:1609.07009.

Simoncelli, E. P. and Olshausen, B. A. (2001). Natural image statistics and neural representation. *Annual review of neuroscience*, 24.

Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. In *Proceedings of the International Conference on Machine Learning*.

Smolensky, P. (1986). Information processing in dynamical systems: Foundations of harmony theory. Technical report, DTIC Document.

Springenberg, J. T. (2016). Unsupervised and semi-supervised learning with categorical generative adversarial networks. In *Proceedings of the International Conference on Learning Representations*.

Sutskever, I. and Tieleman, T. (2010). On the convergence properties of contrastive divergence. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*.

Theano Development Team (2016). Theano: A Python framework for fast computation of mathematical expressions. arXiv:1605.02688.

Theis, L., van den Oord, A., and Bethge, M. (2016). A note on the evaluation of generative models. In *Proceedings of the International Conference on Learning Representations*.

Tieleman, T. (2008). Training restricted Boltzmann machines using approximations to the likelihood gradient. In *Proceedings of the International Conference on Machine Learning*.

Tieleman, T. and Hinton, G. (2009). Using fast weights to improve persistent contrastive divergence. In *Proceedings of the International Conference on Machine Learning*.

Titsias, M. K. and Ruiz, F. J. (2018). Unbiased implicit variational inference. arXiv:1808.02078.

Tran, D., Ranganath, R., and Blei, D. (2017). Hierarchical implicit models and likelihood-free variational inference. In *Advances in Neural Information Processing Systems*.

Ulyanov, D., Lebedev, V., Vedaldi, A., and Lempitsky, V. (2016a). Texture networks: Feed-forward synthesis of textures and stylized images. In *Proceedings of the International Conference on Machine Learning*.

Ulyanov, D., Vedaldi, A., and Lempitsky, V. (2016b). Instance normalization: the missing ingredient for fast stylization. arXiv:1607.08022.

Ulyanov, D., Vedaldi, A., and Lempitsky, V. (2017). Improved texture networks: Maximizing quality and diversity in feed-forward stylization and texture synthesis. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*.

van den Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., W. Senior, A., and Kavukcuoglu, K. (2016a). Wavenet: A generative model for raw audio. arXiv:1609.03499.

van den Oord, A., Kalchbrenner, N., and Kavukcuoglu, K. (2016b). Pixel recurrent neural networks. In *Proceedings of the International Conference on Machine Learning*.

van den Oord, A., Kalchbrenner, N., Vinyals, O., Espeholt, L., Graves, A., and Kavukcuoglu, K. (2016c). Conditional image generation with PixelCNN decoders. In *Advances in Neural Information Processing Systems*.

van Merriënboer, B., Bahdanau, D., Dumoulin, V., Serdyuk, D., Warde-Farley, D., Chorowski, J., and Bengio, Y. (2015). Blocks and Fuel: Frameworks for deep learning. arXiv:1506.00619.

Warde-Farley, D. and Goodfellow, I. (2016). 11 adversarial perturbations of deep neural networks. *Perturbations, Optimization, and Statistics*.

Wei, L.-Y. and Levoy, M. (2000). Fast texture synthesis using tree-structured vector quantization. In *Proceedings of the Conference on Computer Graphics and Interactive Techniques*.

Wu, Y., Burda, Y., Salakhutdinov, R., and Grosse, R. (2017). On the quantitative analysis of decoder-based generative models. In *Proceedings of the International Conference on Learning Representations*.

Yin, M. and Zhou, M. (2018). Semi-implicit variational inference. arXiv:1805.11183.

Younes, L. (1999). On the convergence of markovian stochastic algorithms with rapidly decreasing ergodicity rates. *Stochastics: An International Journal of Probability and Stochastic Processes*, 65.

Zeiler, M. D. and Fergus, R. (2014). Visualizing and understanding convolutional networks. In *Proceedings of the European Conference on Computer Vision*.

Zhao, J., Mathieu, M., Goroshin, R., and Lecun, Y. (2016). Stacked what-where auto-encoders. In *ICLR Workshop Track*.