UNIVERSITY OF AMSTERDAM

MSc Artificial Intelligence

Master Thesis

Semi-Supervised Learning with Generative Adversarial Networks

by

Liam Schoneveld

11139013

September 1, 2017

36 ECTS
February – September, 2017

Supervisor:
Prof. Dr. M. Welling

Assessor:
Dr. E. Gavves

Daily Supervisor:
T. Cohen MSc

Faculteit der Natuurkunde, Wiskunde en Informatica

Abstract

As society continues to accumulate more and more data, demand for machine learning algorithms that can learn from data with limited human intervention only increases. Semi-supervised learning (SSL) methods, which extend supervised learning algorithms by enabling them to use unlabeled data, play an important role in addressing this challenge. In this thesis, a framework unifying the traditional assumptions and approaches to SSL is defined. A synthesis of SSL literature then places a range of contemporary approaches into this common framework.

Our focus is on methods which use generative adversarial networks (GANs) to perform SSL. We analyse in detail one particular GAN-based SSL approach [Dai et al. (2017)]. This is shown to be closely related to two preceding approaches. Through synthetic experiments we provide an intuitive understanding and motivate the formulation of our focus approach. We then theoretically analyse potential alternative formulations of its loss function. This analysis motivates a number of research questions that centre on possible improvements to, and experiments to better understand the focus model. While we find support for our hypotheses, our conclusion more broadly is that the focus method is not especially robust.

Acknowledgements

I would like to thank Taco Cohen for supervising my thesis. Despite his busy schedule, Taco was able to provide me with invaluable feedback throughout the course of this project.

I am also extremely grateful to Auke Wiggers for his mentoring, discussions and guidance. He really helped me to think about these problems in a more effective way.

Tijmen and the rest of the Scyfer team deserve a special mention for providing a working environment that was fun but also set the bar high.

My gratitude also goes out to my committee of Max Welling and Efstratios Gavves for agreeing to read and assess my work among their demanding schedules.

I gratefully acknowledge Zihang Dai, author of the paper which is the central focus of this thesis, for his timely and insightful correspondence via email.

Finally I would like to thank my parents, brother, and my grandpa.

Contents

Part I
Background

1 Introduction

1.1 The scarcity of labels

> *"If intelligence was a cake, unsupervised learning would be the cake, supervised learning would be the icing on the cake, and reinforcement learning would be the cherry on the cake. We know how to make the icing and the cherry, but we don't know how to make the cake."*
>
> Yann LeCun (2016)

Today, data is of ever-increasing abundance. It is estimated that 4.4 zettabytes of digital data existed in 2013, and this is set to rise to 44 zettabytes (i.e., 44 *trillion* gigabytes) by 2020 [International Data Corporation (2014)]. Such a vast store of information presents substantial opportunities to harness. Machine learning provides us with ways to make the most of these opportunities, through algorithms that essentially enable computers to learn from data.

One significant drawback of many machine learning approaches however, is that they require annotated data. Annotations can take different forms and be used in different ways. Most commonly though, the annotations required by machine learning algorithms consist of the desired targets or outputs our trained model should produce after being shown the corresponding input.

It is relatively uncommon that data found 'in the wild' comes with ready-made annotations. There are some exceptions, such as captions added to images by online content creators, but even these may not be appropriate for the particular objective. Manually annotating data is usually time-consuming and costly. Hence, as Yann LeCun's quote suggests, there is an ever-growing need for machine learning methods designed to work with a limited stock of annotations. As we explain in the next section, algorithms designed to do so are known as unsupervised, or semi-supervised approaches.

1.2 Semi-supervised learning

To define semi-supervised learning (SSL), we begin by defining supervised and unsupervised learning, as SSL lies somewhere in between these two concepts.

Supervised learning algorithms are machine learning approaches which require that every input data point has a corresponding output data point. The goal of these algorithms is often to train a model that can accurately predict the correct outputs for inputs that were not seen during training. That is, it learns a function from training data that we hope will *generalise* to unseen data points.

Unsupervised learning algorithms are those which require only input data points – no corresponding outputs are provided or assumed to exist. These algorithms can have a variety of goals; unsupervised generative models are generally tasked with being able to generate new data points from the same distribution as the input data, while a range of other unsupervised methods are focused on learning some new representation of the input data. For instance, one might aim to learn a representation of the data that requires less storage space while retaining most of the input information (i.e., compression).

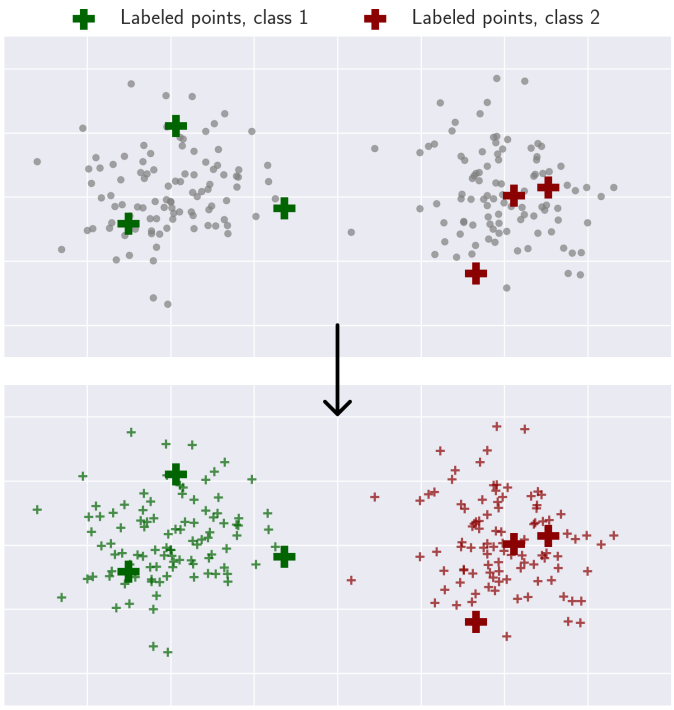


Figure 1: Illustration showing the general idea behind many SSL classification algorithms, based on the cluster assumption. The known labels in the top panel are propagated to the unlabeled points in the clusters they belong to.

SSL algorithms fall in between these paradigms. Strictly speaking, SSL methods are those designed to be used with datasets comprised of both annotated (or labeled) and unannotated (or unlabeled) subsets. Generally though, these methods assume the number of labeled instances is much smaller than the number of unlabeled instances. This is because unlabeled data tends to be more useful when we have few labeled examples. As explained in Section 3.1, in general these methods rely on some assumption of smoothness or clustering of the input data. The main intuition at the core of most semi-supervised classification methods is illustrated in Figure 1.

In the context of today's data-rich world as described in the introduction, we believe that SSL methods play a particularly important role. While unsupervised learning is vital, as we cannot expect to annotate even a tiny proportion of the world's data, we believe SSL might prove equally important, due to its ability to give *direction* to unsupervised learning methods. That is, the two sides of SSL can assist one another with regards to the practitioner's task; the supervised side directs the unsupervised side into extracting structure that is more relevant to our particular task, while the unsupervised side provides the supervised side with more usable information.

2 Overview and contributions

2.1 Overview

This thesis is structured as follows. In Section 3, we present the assumptions that are required for SSL, and the place the main categories of historic SSL approaches into a contemporary context. In Section 4 we synthesise the SSL literature, focusing mainly on approaches that involve deep learning in some way.

Readers that are already well-versed in the basic concepts of SSL, and the surrounding literature, can be advised to skip or skim Sections 3 and 4.

In Section 5 we give a detailed background on three related SSL models. These are all based on generative adversarial networks (GANs) and form a central focus of this thesis. In Section 6 we motivate a number of more basic cost functions used to enable SSL and illustrate their behaviour through synthetic experiments. This exercise gives readers a more intuitive understanding behind the approach taken in our focus model. We then undertake a more theoretical analysis of these loss functions, derive some new alternatives, and hypothesise about their potential advantages and disadvantages in the context of our focus model.

Based on the preceding analysis and discussion, in Section 7 we formulate a number of research questions. Then in Section 8 we address each of these questions through larger empirical experiments, and present and discuss the results. Finally in Section 9 we conclude our study, give some practical recommendations, and suggest promising directions for future research.

2.2 Contributions

The contributions made in this thesis include:

- A broad review of deep learning-based approaches to SSL, placed into a historical context (Sections 3, 4 and 5).
- An intuitive walk-through, that motivates and illustrates the behaviour of a number of loss functions commonly found in SSL models (Section 6.1). This is also used to more clearly explain the logic behind our focus model (the *BadGAN* model, introduced in Section 5.4).
- Theoretical analysis of these loss functions and the introduction of alternative options, alongside a theoretical comparison between the approaches used by the *CatGAN* (introduced in Section 5.2) and *Improved GAN* (introduced in Section 5.3) models (remainder of Section 6).
- Larger empirical experiments, which address the following research questions:

  Which of our analysed loss functions, or approaches to the BadGAN-style model, works best?

  Can the PixelCNN++ component of the BadGAN model be replaced by something that is faster to train and infer from?

  Do the generated images actively contribute to higher-order feature learning of the classifier network in some way?

  Is Virtual Adversarial Training [Miyato et al. (2017)] truly orthogonal to a BadGAN-style approach, as asserted in Dai et al. (2017)?
- The revival of *Information Regularisation*, a SSL approach from 2006, derivations making it suitable for use with neural networks, and evidence suggesting it is competitive with modern approaches (Appendix A).

3 Assumptions and approaches that enable semi-supervised learning

In this section we introduce and explain the main assumptions that have been identified in the literature as enabling SSL. We then go on to describe the main paradigms which SSL methods fall into, and comment on their relevance in a contemporary context.

3.1 Assumptions required for semi-supervised learning

To perform SSL, some assumption must be made about the relationship between labeled and unlabeled data points. In our experience and review of the literature, we have found that the scope of possible assumptions is in fact quite limited. We essentially have the assumption of smoothness (explained in the first subsection below), and possibly in addition, the existence of a manifold where this smoothness assumption is better satisfied. Despite this, these basic assumptions can be interpreted in numerous ways, and have given rise to a wide range of approaches.

3.1.1 Smoothness assumption

There is one key assumption that we need to make in order for SSL to be possible. This is the *smoothness assumption* (Assumption 3.1.1).

Assumption 3.1.1: Smoothness assumption

If two points x_1, x_2 in a high-density region are close, then their corresponding outputs y_1, y_2 should also be close [Chapelle et al. (2006)].

It should be noted that this closeness referred to in the assumption need not exist in the original space of x; it could also only exist in some (possibly non-linear) manifold of the original space. Moreover, the notion that smoothness only needs to hold in high density regions of the data or manifold space embeds a further assumption that points in low-density regions tend to be outliers, and thus the smoothness assumption may not hold for those points.

In SSL, we wish to make inferences about the outputs of our unlabeled data points, by linking them in some way to our labeled data. Without the smoothness assumption, there is no real way of linking our labeled data points to our unlabeled data points (the need for the smoothness assumption is also pertinent in fully-supervised settings, if we are expecting our model to be able to generalise to unseen data points). Fortunately, the smoothness assumption is quite reasonable and we expect it to hold in most real-world applications.

In fact, any problem domain is theoretically smooth if we imagine there is a smooth manifold linking the concepts we are trying to model or classify. Consider for instance a sentiment classifier trained on sentences. We could represent each sentence by a sparse vector whose length is the total number of words in the vocabulary. The vector has zeros everywhere except for the positions corresponding to the words in the sentence, which each contain the occurrence count of that word in the sentence. The space of the sentences in this representation is very non-smooth; similar concepts expressed with different words will in general be no closer to one another than entirely different concepts. However, if we imagine there is some embedding space whose dimensions encapsulate the semantic meaning of a word or sentence (like that which can be obtained through the word2vec algorithm [Mikolov et al. (2013)]) then in this imagined space, smoothness should hold. Thus, so long as we can imagine a smooth manifold linking concepts, then the smoothness assumption holds, provided this manifold exists and we can discover it. Assuming the existence of a discoverable manifold is another assumption, introduced in Section 3.1.4.

3.1.2 Cluster assumption

A cluster can be defined as a densely-populated region of our data that can be connected by short curves that do not pass through any low-density regions [Chapelle et al. (2006)]. The cluster assumption (Assumption 3.1.2) has been described as being used by "virtually all successful semi-supervised algorithms" [Chapelle and Zien (2005)].

Assumption 3.1.2: Cluster assumption

If two points x_1, x_2 are within the same cluster, then their corresponding outputs y_1, y_2 should also be similar [Chapelle et al. (2006)].

Given the smoothness assumption, if we also assume the presence of distinct classes, then we can expect the data to form clusters, to some extent. Specifically, if our outputs y_1, y_2 are classes, then under the smoothness assumption we expect points x_1, x_2 to be close to one another if they have the same class label, and distant from one another if they have different class labels. From this it follows that we can expect the points in $\mathcal{X}$ to form clusters (where $\mathcal{X}$ is defined as the set of all possible input data points, of which x_1 and x_2 are a subset).

However, the assumption that items in the same cluster should generally be of the same class does not imply that the items from a particular class will form only one cluster. For instance, if we consider the MNIST handwritten digits dataset [LeCun (1998)], in the data's original pixel space the classes will form many small clusters. The clusters might represent, for instance, *'thin eights slanting to the right,'* or *'large threes.'* Such a clustering would of course be only somewhat useful for our classification task, particularly when we only have a small amount of labeled data. We would likely prefer to learn some manifold that has a smaller number of clusters (but also such that items from different classes remain in distinct clusters).

3.1.3 Low-density separation

Another way of arriving at the cluster assumption is by assuming that the classes are separated by regions of low density [Chapelle et al. (2006)]. Since any high-density region must be part of a cluster, separating the classes within such a region would violate the cluster assumption (as the outputs would then differ for multiple data points within a single cluster).

Assumption 3.1.3: Low-density separation assumption

The boundaries between classes should lie in low-density regions of the input space [Chapelle et al. (2006)].

Like the cluster assumption, the low-density separation assumption arises automatically once we assume smoothness and the presence of classes. However, thinking about the data or feature space in terms of one that is clustered, versus one that is divided by regions of low-density, can inspire different algorithms. For instance, thinking in terms of clusters intuitively suggests a solution where we assume the data-generating distribution is a mixture of Gaussians, wherein instead of being completely latent, the mixing probabilities are known for our labeled data points [Zhu (2007)]. On the other hand, viewing the space from a low-density separation perspective might inspire an algorithm where we use the unlabeled data to approximate where the high-density regions of our input space lie, and then introduce an aspect to our model that somehow uses this information to encourage the class decision boundaries to be placed outside these regions (this approach is taken in Corduneanu and Jaakkola (2006), for example).

Note that the cluster or low-density separation assumptions do not imply that there is necessarily just one cluster per class. Rather, the number of clusters may lie anywhere on

the spectrum from one cluster per class, to the extreme case of one 'cluster' per datapoint. However we generally assume that the latter extreme will be rare in practice.

3.1.4 Existence of a discoverable manifold

In many cases, the above assumptions are hypothesised to only hold in some (usually lower-dimensional) manifold of the original data space. In such a case, to apply SSL we must assume that such a manifold exists (Assumption 3.1.4).

Assumption 3.1.4: Manifold existence assumption

There exists some manifold that our data can be projected onto and that provides a more useful representation of our data [Chapelle et al. (2006)].

If we make this assumption, then we must also have a way of projecting our data onto this manifold. Presumably we must learn such a projection – otherwise we could just use the manifold representation as our starting point. Thus we generally also assume here that there is a way to learn the projection to the assumed manifold.

The prominence of deep neural networks has made this assumption increasingly relevant, as deep learning is centred on the idea of learning intermediate layers of features, which can be seen as lower-dimensional manifolds which we transform our data onto.

3.2 Classes of semi-supervised learning algorithms

In Chapelle et al. (2006), four classes of SSL methods are introduced. Though this reference dates to 2006, we believe its typology is still appropriate in a contemporary context. In this section, we briefly introduce each of these classes, and provide comments on their relevance in the modern machine learning environment, where algorithms that employ deep neural networks are emerging as a dominant force across a number of applications.

3.2.1 Methods based on generative models

Generative models enable inference by explicitly modelling $p(x|y)$ and then using Bayes' rule to obtain $p(y|x)$. In this setting, any additional information about $p(x)$ is directly useful. Thus generative methods would seem to lend themselves well to SSL.

The mixture of Gaussians example alluded to in Section 3.1.3 is a key example of this. In that example, the input data points x are assumed to have been generated according to $p(x) = \sum_y p(x|y)p(y)$, where each $p(x|y)$ is an independent Gaussian distribution. We can estimate the parameters of this mixture of distributions using the Expectation Maximisation algorithm. Since we know some of the labels y, we can incorporate this information into the parameter estimation procedure by treating not all labels as latent variables.

The generative paradigm generated a lot of interest during earlier periods of SSL research [Chapelle et al. (2006)]. Despite this, these methods have never shown themselves to be especially effective. We believe this is due to the requirement for the $p(x)$ distribution to be fully specified and estimated. This is problematic for two reasons:

1. It requires us to make often-restrictive or simplistic assumptions over the functional form of $p(x)$, such as that it is a mixture of Gaussians.
2. The requirement of estimating a complete model for $p(x)$ means we are directing substantial modelling capacity towards estimating something that is only indirectly related to our end goal of estimating $p(y|x)$.

It is due to these limitations that we believe generative approaches have not been shown to perform so well, and as a result have received decreasing levels of attention in SSL

literature.[1]

3.2.2 Methods based on low-density separation

This category encapsulates all methods whose central goal is for the model's decision boundaries to be placed in low-density regions of the input data space. The low-density separation assumption is much less restrictive than the generative approach, as the goal of having model decision boundaries being placed in low-density regions can be achieved in many different ways. As such, algorithms based on this assumption are widespread and diverse. This flexibility has seen these approaches remain relevant in the era of deep learning. We do not address this category further here as much of the literature reviewed in the next section falls under this category, and as these approaches are also the central focus of later parts of this thesis.

3.2.3 Graph-based methods

In graph-based approaches to SSL, the data are assumed to form a graph. Input data points are the nodes of the graph, and the edges are generally some measure of distance between these points. The adjacency matrix of this graph can then be used in some way to determine how similar the unlabeled data are to one another, and to the labeled data. This can then be used to make predictions for the unlabeled nodes, or to provide additional information to (i.e., regularise) a classifier.

Graph-based approaches that are used purely to produce labels for the unlabeled nodes of the graph fall under a category known as *transductive* approaches. As opposed to inductive approaches, which learn a function capable of estimating the label for any given input data point, transductive approaches only produce labels for the fixed set of unlabeled data that was used during training.

In the mid-2000s, graph-based and transductive methods were the most active area of SSL research [Chapelle et al. (2006)]. A decade on, such methods are not as prominent, as they do not lend themselves so readily to deep learning-based approaches. That said, as we describe in Section 4, some of the deep learning-based approaches we review could be viewed as graph-based, or at least inspired by a graph-based perspective of the data.

3.2.4 Methods based on a change of representation

In Chapelle et al. (2006), the authors separate out a category of SSL approaches that require two steps. In the first step, a new representation of the data is learned in an unsupervised manner. The labeled data points are then transformed into this new representation, and a fully-supervised classifier is trained on the transformed data.

This highlights a fundamental shift brought on by deep learning. In more modern deep learning-based approaches, such a change of representation is integral, as at their core these approaches are about learning intermediate layers of features that represent the input data in a more meaningful or useful way. The above-mentioned notion of 'two-step' learning referred to in Chapelle et al. (2006) however, is increasingly uncommon. Neural networks instead leverage their ability to learn both a new data representation, and a classifier on that representation, at the same time. The ability of neural networks to do so has generally meant that the representation learned is more useful for the classification task, as the training signal from that task is already present during representation learning. Thus, while this category is still very relevant, its precise definition can be seen as having changed somewhat in the ten or so years following its introduction in Chapelle et al. (2006).

[1] While some more recent deep learning-type SSL models are generative in the sense that they learn models that are capable of generating new data points (for example, methods that use GANs or variational autoencoders), these approaches generally do not explicitly provide an analytical expression for $p(x|y)$, and thus do not fall under the specific type of generative approach we refer to here.

Part II
Related Work

4 Review of Semi-Supervised Learning Literature

In this section we provide an overview of literature pertaining to SSL. Though the focus of this section is more on recent deep learning-based methods, we begin by briefly describing some of the more historic approaches to SSL. In the final subsection we turn our focus to SSL methods based on GANs Goodfellow et al. (2014)]. Before proceeding, in Box 1 we introduce three datasets that frequently are used as standard benchmarks for performance in the SSL algorithms literature.

Box 1: Benchmark datasets commonly referred to throughout this thesis

To compare and evaluate the SSL methods presented throughout this thesis, we frequently refer to three key benchmarks. These benchmarks are standard datasets, with a standard configuration of the number of labeled examples.

The MNIST dataset [LeCun (1998)] consists of greyscale 28 by 28 pixel images of individual handwritten digits, labeled from 0 to 9. It contains 70,000 handwritten digits (60,000 training images and a fixed test set of 10,000 images containing 1,000 images from each class). **PI-MNIST-100** refers to a version where all but 100 labels (10 per class) are randomly censored for the particular training run. PI means 'permutation invariant,' which essentially means that the applied model must still work were the pixels in each MNIST digit randomly reordered (with the same permutation applied to each image in the dataset). As such, approaches like convolutional neural networks lose their effectiveness in the PI context, as they rely upon the spatial structure of the inputs. So when referring to PI-MNIST, we effectively mean that only multilayer perceptrons (MLPs) can be used, and not CNNs.

CIFAR10-4k refers to the CIFAR-10 dataset [Krizhevsky (2009)], with all but 4000 labels (400 per class) randomly censored for the particular training run. CIFAR-10 is a dataset of 70,000 red-green-blue (RGB) images of size 32 by 32 pixels (the 70,000 images are also split into 60,000 training images and 10,000 test images). The images contain one of and are labeled into ten categories: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck.

SVHN-1k refers to the SVHN dataset [Netzer et al. (2011)], with all by 1000 labels (100 per class) randomly censored for the particular training run. The SVHN dataset consists of 32 by 32 RGB images of house numbers taken from *Google StreetView*^{TM}. Though the images can contain multiple digits, the labels are from 0 to 9, and refer to the digit at the centre of each image. There are 73,257 images in the training set and 26,032 images in the test set.

4.1 Semi-supervised learning before the deep learning era

SSL first came to prominence during the 1970s. Possibly the earliest conceived approach is known as 'self-learning,' an iterative procedure where some unlabeled points (usually those with the most confident predictions) are labeled with the current supervised model's best prediction, thereby providing more training data for the supervised learning algorithm. Blum and Mitchell (1998)'s co-training offers a similar approach, wherein two models are trained on two separate subsets of the data features. Confident predictions from one model are then used as labeled data for the other model. A large focus for SSL during the 1970s

was on generative models [Chapelle et al. (2006)], which we described in Section 3.2.1.

Transductive Support Vector Machines (TSVMs) [Joachims (1999)] are another popular technique that rose to prominence during the 'SVM boom' of the late 1990's. Like regular SVMs, TSVMs aim to maximise the margin between the decision boundary and the data points. TSVMs then extend this by additionally taking into account the distance of unlabeled data to the margin, and attempting to simultaneously maximise this. This semi-supervised version of the SVM algorithm is more difficult to solve as it is non-convex, however approximate algorithms do exist [Chapelle et al. (2006)].

Other SSL methods take a graph-based approach. Some similarity measure between samples is required, and these form the edges of the graph. The nodes are then our unlabeled and labeled data points. In Section 4.2.3 we describe one neural network-based SSL method that implicitly uses a graph structure. One example of a graph-based method that pre-dates deep learning is *label propagation* [Zhu and Ghahramani (2002)], which aims to minimise the difference between predicted labels for nodes with heavily weighted edges. The result of this process is that label information propagates out to nearby unlabeled points.

4.2 Semi-supervised deep learning

Over the decade after around 2007, deep learning, which generally refers to the use of neural networks containing more than one hidden layer, became increasingly dominant as a preferred machine learning method. In the latter half of that decade, the popularity of deep learning exploded, driven primarily by large performance gains across a range of learning tasks due to improved algorithms and hardware. SSL has not been exempt from these performance gains; the deep learning 'revolution' has led to similar gains in performance across many SSL benchmarks.

In fact, arguably deep learning's impact on SSL methods is even more marked. Using image classification as the canonical example, semi-supervised approaches reap all of the benefits brought on by deep learning, such as the efficacy of convolutional neural networks (CNNs) to learn from image data, and the effectiveness of the notion of learning intermediate layers of features.

In addition to this however, SSL also benefits from the modularity of deep learning. Neural networks allow for greater flexibility over the ways in which labeled and unlabeled data can be used together to train a model. Thus we believe that the proportional improvement afforded by deep learning on SSL methods is likely to be even greater than that for supervised-only methods.

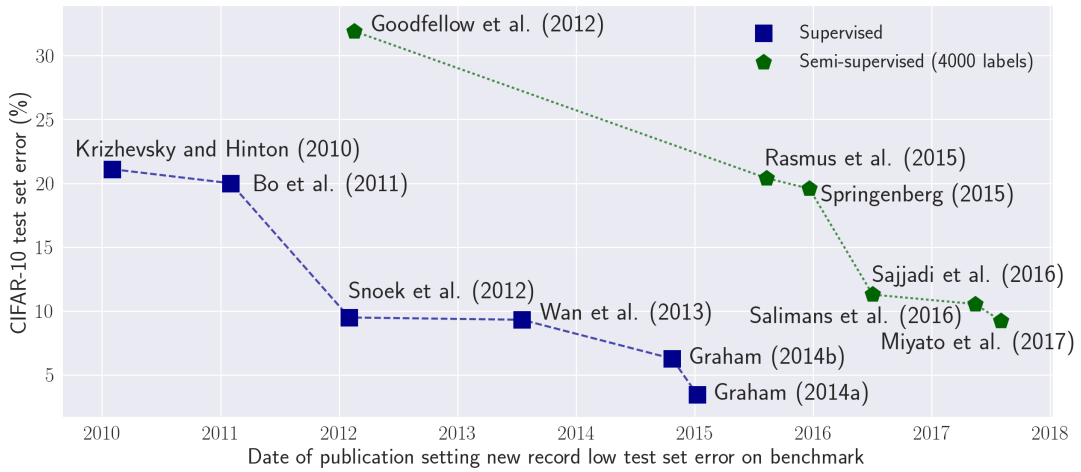


Figure 2: Progress of performance in fully-supervised (50,000 labels) and SSL (4,000 labels; 50,000 data points) on the CIFAR-10 dataset over time. While SSL performance has continued to improve since 2012, progress on the fully-supervised task has slowed.

A comparison (Figure 2) of the progress of performance over time on the CIFAR10-4k semi-supervised classification benchmark versus that on fully-supervised CIFAR10 supports this notion. For the fully-supervised benchmark, the impact of deep CNNs is clear, with a drop in error rates of around ten percentage points being afforded in 2012. Since then, progress on the fully-supervised benchmark has slowed, while at the same time, improvements in performance have continued to occur on the semi-supervised benchmark. We believe this is due to researchers beginning to realise the additional capabilities of deep learning-based approaches when applied in SSL contexts.

In this section we present some of the most prominent and effective neural network-based approaches to SSL. These approaches are grouped into broad categories based on similarities in their general approach. Per the focus of this thesis, particular attention is given to SSL methods that make use of GANs in some way. In Box 2 we present the general notation for SSL with a neural network that will be used in this section and throughout this thesis.

Box 2: Notation used to describe neural network classifiers in this thesis

Many of the models discussed in the literature review, as well as all of those described in Part III, consist of a neural network classifier with a specific loss function. Here we present the standard notation used throughout this thesis to describe such networks.

Assume we have a dataset:

- Comprised of N feature vectors (data points), forming the rows of matrix $\mathbf{X}$
- Whose first N_L points $\mathbf{X}_L = \{\mathbf{x}_1, \ldots, \mathbf{x}_{N_L}\}$ have known corresponding class labels $\mathbf{y}_L = \{y_1, \ldots, y_{N_L}\}$
- And whose remaining $N_U = N - N_L$ points $\mathbf{X}_U = \{\mathbf{x}_{L+1}, \ldots, \mathbf{x}_N\}$ are unlabeled.

Our goal is to learn a classifier function $C(k|\mathbf{x}, \boldsymbol{\theta})$ that takes as input a data point $\mathbf{x}$ and outputs a probability distribution over label values $k \in \{1, \ldots, K\}$ based on some parameters $\boldsymbol{\theta}$.

To achieve this goal we can construct a functional $F(C(\cdot|\mathbf{x}, \boldsymbol{\theta}); \mathbf{X}, \mathbf{y})$, which evaluates the efficacy of a particular $C(\cdot|\mathbf{x}, \boldsymbol{\theta})$. To exemplify this, we present one such functional, the categorical *cross-entropy loss*, which appears extensively throughout this thesis:

$$\mathcal{L}_{\text{xent}} = \mathbb{E}_{\mathbf{x}, y \sim p_L(\mathbf{x}, y)} \left[\sum_{k=1}^{K} \delta(y = k) \log C(k|\mathbf{x}) \right] \tag{4.1}$$

where δ is the Dirac delta (or indicator) function, and we have omitted C's dependence on $\boldsymbol{\theta}$ for brevity, which we often do throughout this thesis.

In Equation 4.1 $p_L(\mathbf{x}, y)$ represents the true distribution of the labeled data points (we similarly use $p_U(\mathbf{x})$ to represent the unlabeled data distribution and $p(\mathbf{x})$ to represent the overall distribution of $\mathbf{x}$). This 'expectation form' of loss components is used throughout the rest of this thesis as the general form for presenting neural network loss components.

We do not know the true p_L distribution, so when training a neural network in practice we approximate the expectation in Equation 4.1 by Monte Carlo sampling from the labeled data in minibatches, and then performing stochastic gradient descent on these minibatches. An equivalent approach can be assumed to be taken for all expectation-based loss terms appearing throughout this thesis.

4.2.1 Autoencoder-based approaches

In large part, deep learning grew out of unsupervised methods like Deep Belief Networks, which are generally comprised of unsupervised methods such as Restricted Boltzmann Machines, or autoencoders [Hinton (2009)]. These algorithms were already unsupervised, and often were used to extract features from data prior to training a classifier with these features. Such approaches naturally lent themselves to SSL as the unsupervised pre-training and supervised classification training stages were already decoupled; one could simply perform the latter stage with just the smaller, labeled subset of the data. Due to this, many of the early approaches to using neural networks for SSL involved autoencoders[2] in some way.

An early approach to SSL with neural networks that made use of autoencoders was **Pseudolabel** [Lee (2013)]. Pseudolabel is essentially self-learning; the model is trained on

[2] An *autoencoder* is a neural network that is trained to reconstruct its input. Usually there is a bottleneck at the centre of such a network, so that a compressed representation of the input must be learned. Often, noise is added to corrupt the input, and the task is to reconstruct the non-corrupted input; this case is referred to as a *denoising autoencoder*.

Approach	MNIST	SVHN	CIFAR10
TSVM [Joachims (1999)]	16.81	66.55	
Pseudolabel [Lee (2013)]	10.49		
FastInfoReg[1]	1.74		
Auxiliary Deep Generative Models[2]	0.96	16.61	
Ladder Network [Rasmus et al. (2015)]	1.06		20.40
CatGAN [Springenberg (2015)]	1.91		19.58
Improved GAN [Salimans et al. (2016)]	0.93	8.11	18.63
Triple GAN [Li et al. (2017)]	0.91	5.77	16.99
Kumar et al. (2017)		5.90	16.78
Learning by Association[3]		5.14	
VAT + Entropy Minimisation[4]	1.08	4.81	
BadGAN [Dai et al. (2017)]	**0.80**	**4.25**	14.41
VAT [Miyato et al. (2017)]	1.36	6.83	12.36
Temporal Ensembling[5]		4.42	12.16
VADropout [Park et al. (2017)][6]			11.32
Sajjadi et al. (2016)		6.03*	11.29
VAT + VADropout[6]			**9.22**

Table 1: Performance (% error) of various SSL algorithms on the PI-MNIST-100, SVHN-1k and CIFAR10-4k datasets. Ordering priority is CIFAR10, then SVHN, then MNIST.

Notes:
1: Our adaptation of Corduneanu and Jaakkola (2006).
2: Maaløe et al. (2016). Does not use convolution neural networks for SVHN.
3: Haeusser et al. (2017).
4: Our implementation.
5: Laine and Aila (2016).
6: Uses larger CNN architecture for CIFAR10 than other models (which all use the same smaller architecture).

the labeled and unlabeled data, and prior to each iteration the unlabeled data is temporarily labeled with the model's current most confident prediction. That is, Pseudolabel trains a neural network with the loss function $\mathcal{L}_{\text{pseudolabel}}$, where:

$$\mathcal{L}_{\text{xent/argmax}} := \mathbb{E}_{\mathbf{x}\sim p_U(\mathbf{x})}\left[\sum_{k=1}^{K}\delta\left(\underset{k}{\text{argmax}}\ C(k|\mathbf{x}) = k\right)\log C(k|\mathbf{x})\right] \tag{4.2}$$

$$\mathcal{L}_{\text{pseudolabel}} := \mathcal{L}_{\text{xent}} + \alpha(t)\mathcal{L}_{\text{xent/argmax}} \tag{4.3}$$

where $\alpha(t)$ schedules the amount of regularisation penalty to be added to the loss. The authors assert [Lee (2013), Section 3.2] that this approach is equivalent to entropy minimisation [Grandvalet and Bengio (2005)], however we show in Appendix B that these approaches are in fact not entirely equivalent.

The authors find that Pseudolabel improves performance on PI-MNIST-100 over a neural network with dropout [Srivastava et al. (2014)]. They further improve this performance by sharing their classifier's features with those of a denoising autoencoder whose reconstruction cost serves as an additional training signal (Table 4.2.1).

Ladder Networks [Rasmus et al. (2015)] are another SSL approach, first introduced in 2015 with competitive performance (Table 4.2.1). Figure 3 sketches the overall design, and we temporarily switch to that diagram's notation here. The starting point is essentially an autoencoder. Noise is added at every layer in the encoder path, creating a 'corrupted' pass (h is the clean pass, z is the corrupted pass). A 'clean' pass through this encoder of a batch of unlabeled data (indicated by $\tilde{x}$, while x represents the labeled data) is also made at every training iteration. The final layer of the encoder has as many units as the number of classes. For the corrupted pass of the labeled examples this output is sent through a softmax to give class probabilities and a standard cross-entropy loss term is calculated.

For the unlabeled batch, the final layer activations are then sent back through a decoder. The task of the decoder is to reconstruct the clean activations in a layer-wise fashion. To achieve this, each layer has a denoising function. This receives as input the estimate of the clean version from the layer below, plus, through a lateral connection, the corrupted version of the layer activations it is trying to reconstruct. Batch normalisation [Ioffe and Szegedy (2015)] is added at each layer in a specific ordering when passing through the encoder, and this operation is inverted when going back through the decoder.

Pezeshki et al. (2016) perform an ablation study of the ladder network to determine which of its many components is most responsible for its strong performance. They find that the lateral connections are of most importance, as well as the corruption process. They also find that performance can be slightly improved by increasing the complexity of the denoising functions.

A number of related methods make use of variational autoencoders (VAEs) [Kingma and Welling (2013)] for SSL. One effective (Table 4.2.1) example of this is **Auxiliary Deep Generative Models** [Maaløe et al. (2016)], which introduces an auxiliary latent variable a to the standard VAE setup. This variable is intended to extract features from a given input $\mathbf{x}$ that are particularly informative about the class of that input. A stronger training signal for the auxiliary variable's inference model is achieved by sharing the auxiliary inference model $q_\phi(a|x)$ between both the generative model $p(x|z)$ and inference model $p(z|x)$. With this setup, a variational lower bound on the joint distribution $p(x, y)$ can be maximised for labeled examples, as well as for the distribution $p(x)$, by marginalising out y.

4.2.2 Regularisation and data augmentation-based approaches

A large subset of SSL methods incorporate the notion of learning a classifier that is regularised by a loss penalty term that is derived from the unlabeled data in some way. These types of methods are the focus of the experiments and analysis presented later in this thesis. One key approach is **Entropy Minimisation** [Grandvalet and Bengio (2005)], which

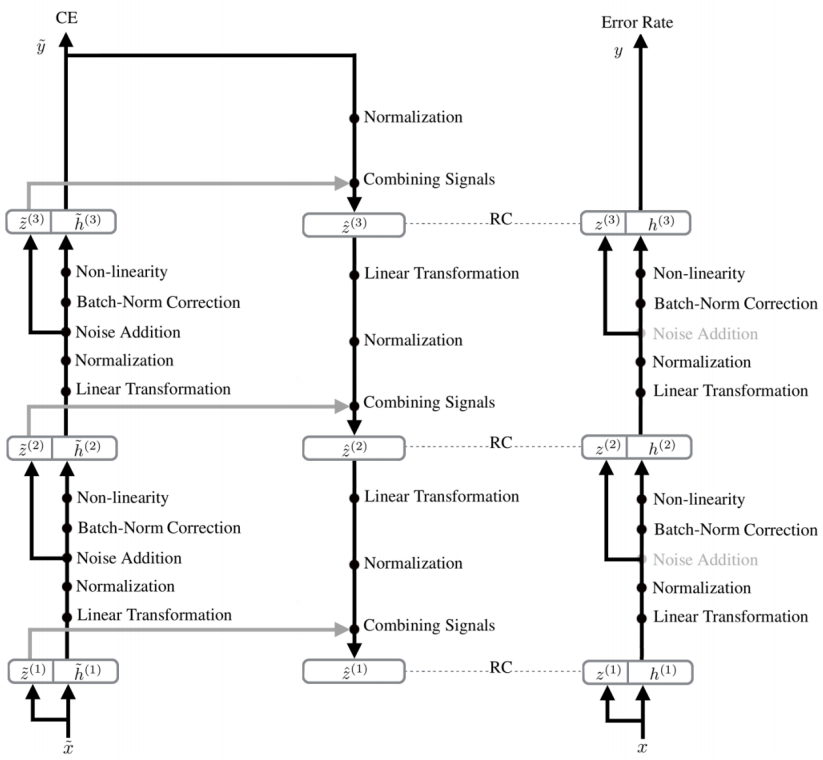


Figure 3: Schematic diagram of the Ladder Network [Rasmus et al. (2015)], from Pezeshki et al. (2016). The corrupted pass through the encoder proceeds up the left side of the diagram. The decoder pass runs from top to bottom through the centre of the diagram. The clean pass through the encoder is to the right.

encourages low-density separation (i.e., placing decision boundaries in low-density regions) by essentially telling the model *not* to put decision boundaries in *high*-density regions. Another is **Regularised Information Maximisation** [Krause et al. (2010)], which adds an entropy maximisation term on the average probability distribution predicted by the model to encourage even coverage of all classes, as well as L2 regularisation.

Information regularisation (hereafter, *InfoReg*) is a regularisation technique introduced in Corduneanu and Jaakkola (2006) that can be used to enforce the low-density separation assumption in SSL tasks. Though this 2006 technique pre-dates the more recent resurgence in the use of neural networks, we believe it is still pertinent. In Appendix A we show that, after adapting it for use with neural networks, it remains a competitive method for performing SSL (see also Table 4.2.1).

InfoReg is based on the notion of dividing the input space into overlapping sub-regions. If each region has a 'best' prediction for the output label (say, the average label of the labeled examples in that region), then InfoReg states that predictions for individual points within that region should not deviate too far from this best average prediction. Through this mechanism, InfoReg encourages smoothness in the prediction function.

This idea of overlapping regions is taken to the limit of having an infinite number of regions. In this limit, each region's size approaches zero and the amount of overlap between all regions approaches 100% [Corduneanu and Jaakkola (2006)]. Assuming the joint distribution over inputs and labels can be defined as $Q(x, y) = Q(y|x)P(x|R)$, where R is one of the overlapping regions, then the regulariser – derived by taking the Kullback-Leibler (KL) divergence between the prediction function, $Q(y|x)$, and the 'best' prediction for the region, $Q(y|R)$ – can be defined as the mutual information between x and y, given the region. The authors show that an expression for this regulariser, as the number of regions becomes infinite, reduces to:

$$J(p) = \int p(\mathbf{x})\mathbf{tr}(F(\mathbf{x}))d\mathbf{x} \tag{4.4}$$

Where $\mathbf{tr}$ is the trace operator, and $F(\mathbf{x})$ is the Fisher Information [Ly et al. (2017)], given as:

$$F(\mathbf{x}) = \mathbf{E}_{C(k|\mathbf{x})}\left[(\nabla_{\mathbf{x}} \log C(k|\mathbf{x})) \cdot (\nabla_{\mathbf{x}} \log C(k|\mathbf{x}))^T\right] \tag{4.5}$$

The penalty $J(p)$ reflects the data density at the decision boundary. Thus, by adding this penalty to a model we encourage the decision boundary to be 'pushed out' to low-density regions, and through this bring about low-density separation. Corduneanu and Jaakkola (2006) show in small experiments that their approach achieves this desired effect. As mentioned, in Appendix A, we derive how this method can be used within a neural network for SSL, and illustrate its effectiveness through synthetic and empirical experiments.

Virtual Adversarial Training (VAT) [Miyato et al. (2017)] is a 2017 approach that led to substantial improvements in performance on the CIFAR10 and SVHN benchmark datasets in a SSL setting. Given a neural network, the idea in VAT is to find the optimal adversarial perturbation $\mathbf{r}$ of a real data input $\mathbf{x}$ such that $||\mathbf{r}||_2^2 \leq \epsilon$ and the KL divergence between the model's output of the original data point, $C(\cdot|\mathbf{x})$, and that of the perturbed data point, $C(\cdot|\mathbf{x} + \mathbf{r})$, is maximised. Since this KL divergence loss does not require any label, this approach can be applied in a semi-supervised context by using this additional training signal on the unlabeled data points. The main challenge with the suggested approach in VAT is finding an optimal adversarial perturbation in an efficient manner during training of the neural network. The authors devise a stochastic approximation that they assert works quite well, and their results (Table 4.2.1) suggest that this is indeed the case.

We believe that the process by which VAT enables SSL is likely similar to that of InfoReg. Intuitively, both of these methods penalise strong gradients of the output with respect to the input in regions of high density. InfoReg does this directly (Equation 4.4), while VAT stochastically approximates this by finding perturbations of inputs in high-density regions

that maximally change the output. In other words, VAT finds places in high-density regions where the model output is quite sensitive to the input, and then regularises the model by reducing this sensitivity. Through this mechanism, it similarly encourages low-density separation.

Virtual Adversarial Dropout [Park et al. (2017)] takes the idea of VAT and applies it to dropout. That is, the authors devise an algorithm capable of finding the optimal 'adversarial dropout mask,' such that changing the random initial dropout mask to this mask leads to a sharp change in decision. Again the model is then regularised by a penalty which asserts this sharp change should not occur. Combining this with VAT, the authors achieve further improvements on semi-supervised CIFAR10 (Table 4.2.1).

Another effective approach is introduced in **Regularization With Stochastic Transformations and Perturbations for Deep Semi-Supervised Learning** [Sajjadi et al. (2016)]. The notion is inspired by other approaches (e.g. Dosovitskiy et al. (2014)) that use data transformations that should not change the semantic classification of the input (such as stretching or translating) as data augmentation. In doing so, these approaches attempt to learn a model that is invariant to such transformations. In this particular version a minibatch consists of multiple perturbations of the same input and of the model that input is passed through. The input perturbations include things like translation, shearing and stretching, while the model perturbations refer to different dropout masks being applied. A regularisation penalty is then added to the model which penalises differences in the outputs resulting from these various model and input perturbations.[3] A 'mutual exclusivity' penalty is also added, which discourages trivial solutions and is similar to entropy minimisation. As at writing, this model achieves competitive performance on CIFAR10-4k and SVHN-1k (Table 4.2.1).

A common thread across the above-described methods is invariance. VAT encourages its model to be invariant to small perturbations in the input space. *Regularization With Stochastic Transformations and Perturbations* is also focused on encouraging invariance. It would appear that one of the major shortcomings of neural networks in their raw form is that they do not automatically learn such invariance; rather it needs to be encouraged through regularisation terms or enforced through specific architectures. The success of convolutional neural networks is itself evidence of this; their success owes in large part to their embedding of the prior knowledge that the semantic meaning of images should be translation-invariant. Much other work in the SSL literature and beyond supports this notion of the importance of invariance, one key example being *Group Equivariant Convolutional Networks* [Cohen and Welling (2016)], which achieves supervised-only performance on CIFAR10-4k that is on par with some of these semi-supervised methods by embedding invariance (or more precisely, equivariance) to groups of transformations (such as rotation) into the model.

4.2.3 Other approaches

As described in Section 3.2.4, one common SSL approach is to first learn a new representation of the data, and then train a fully-supervised classifier on the labeled data in this new space. Deep learning has enabled a number of new approaches to such unsupervised representation learning.

One effective example of unsupervised representation learning is given in Doersch et al. (2015), where the authors train a model to predict the relative position of randomly-sampled image patches. For instance, the model might randomly be shown an image patch containing the wheel of a bus, and another patch with its roof – the model must then deduce that the roof of the bus should be above the wheel. The authors argue that this is likely to be an effective means of learning semantically-meaningful representations of image patches. They find that these representations can then be used effectively for various supervised tasks with

[3] Regularisation encouraging invariance to varying dropout masks was in fact first introduced in Bachman et al. (2014), so Sajjadi et al. (2016) can also be viewed as an extension of that work.

only a few labels.

Temporal Ensembling for Semi-Supervised Learning [Laine and Aila (2016)] is another SSL technique which performs well on standard benchmarks (Table 4.2.1). The central idea of this approach is inspired by the observations in Hinton et al. (2015), that so called "soft targets," derived from the predictions made by ensembles of larger neural networks, can provide an effective additional training signal for a smaller network.

In temporal ensembling, as the model trains, its predictions on unlabeled data points are stored into moving averages. These moving averages can then serve as targets for the network on its next pass through the unlabeled dataset (i.e., during the next epoch). Crucially, each epoch is characterised by different random data augmentation and dropout masks being applied. Thus through this method, the network is also regularised to produced similar outputs in response to different augmentations and dropout masks. As such, this method is similar to that of Sajjadi et al. (2016).

Learning by Association [Haeusser et al. (2017)] is another example of how creative uses of the capabilities of deep learning can lead to effective new SSL methods. It also performs strongly on SSL benchmark tasks (Table 4.2.1). In this approach, we pass a batch of labeled and a batch of unlabeled data through the network and obtain the data representations in the feature space (i.e., the activations of the penultimate hidden layer of the neural network). We desire labeled images with the same label to have similar feature representations. We also wish to associate unlabeled images with labeled images. To achieve this, the authors introduce a *walk cost*. The dot product of each feature vector with each other feature vector is taken – this Gram matrix thus reflects the similarity of all possible pairs of feature vectors. This matrix can be written as:

$$\mathbf{M} = \begin{bmatrix} P_{LL} & P_{LU} \\ P_{UL} & P_{UU} \end{bmatrix}$$

where each of these four entries is itself a matrix, where for instance P_{LU} has entries $m_{lu} = \mathbf{f}_l^T \cdot \mathbf{f}_u$, where $\mathbf{f}_l$ is the feature vector (penultimate layer activations) resulting from putting the lth labeled example of that minibatch through the network, and $\mathbf{f}_u$ is the feature vector of the uth unlabeled example.

We then take the softmax over the columns of P_{LU}, obtaining p_{LU}. Each row of p_{LU} can then be seen as representing the probability of 'walking' from any example in the labeled batch to any example in the unlabeled batch. We can similarly obtain p_{UL}, then use $p_{LU} \cdot p_{UL}$ to give the probability of walking from each labeled example to each other labeled example, via any of the unlabeled examples. The model's cost $\mathcal{L}_{\text{walk}}$ is then the cross-entropy of the $L * L$ matrix $p_{LU} \cdot p_{UL}$, and an indicator matrix E with entries $\mathbb{E}_{ij} = 1$ iff the label of example i is equal to that of example j, and 0 otherwise. This is added to a 'visit' cost, which avoids trivial solutions by ensuring the model visits all unlabeled examples reasonably frequently, as well as the standard cross-entropy cost for the labeled examples $\mathcal{L}_{\text{xent}}$.

As a result of this setup, the feature space representations of unlabeled examples are encouraged to 'sit in-between' those of labeled examples. Or equivalently, we can view the unlabeled examples in this context as linking the labeled examples together. In this sense, this approach can also be seen as a graph-based approach to SSL, as we create a graph reflecting the distance between the data points, and then optimise over aspects of this graph. A key difference though is that the edge distances in the graph are not fixed; here we adjust the edge distances to obtain an 'ideal' graph, rather than propagating label information over some graph with fixed edge distances (like, for example, in Zhu and Ghahramani (2002)).

4.3 Semi-supervised generative adversarial networks

In this section we introduce GANs. We define four broad ways in which GANs have been used for SSL. We then review several papers in this space, categorising each according to which of the broad approaches it adopts.

4.3.1 Generative adversarial networks

GANs [Goodfellow et al. (2014)] are an unsupervised learning algorithm that can learn to generate samples from the distribution of the provided dataset. GANs achieve this through the dynamics of an adversarial game, where the parameters of a discriminator network and generator network are updated in a turn-wise fashion.

The discriminator output is intended to reflect the probability that a given sample came from the true data distribution. Its goal is to maximise its output $D(\mathbf{x})$ for real samples and minimise it for generated samples. The generator's learning goal is then to fool the discriminator; it tries to maximise the discriminator's output for its fake samples. Intuitively, this maximisation will push the generator to produce samples that resemble real images. This in turn will force the discriminator to adapt to these more realistic fake samples, and so on.

Formally, the training procedure consists of optimising over following minimax game:

$$\max_{G} \min_{D} \; -\mathbb{E}_{\mathbf{x}\sim p_U(\mathbf{x})}\left[\log(D(\mathbf{x}))\right] - \mathbb{E}_{\mathbf{z}\sim p_{\mathbf{z}}(\mathbf{z})}\left[\log(1 - D(G(\mathbf{z})))\right] \tag{4.6}$$

Here the $\mathbf{z}$ are intended to be a latent representation of the generated images. $p_{\mathbf{z}}(\mathbf{z})$ is some simple distribution that we can sample from (usually Gaussian- or uniform-distributed noise in around 100 independent dimensions). In practice, the expectations in Equation 4.6 are approximated through minibatch Monte Carlo sampling.

Given a generator and the distribution which it defines p_g, the optimal discriminator can be found at the Nash equilibrium [Nash (1951)] of the game, which occurs when:

$$D(\mathbf{x}) = \frac{p(\mathbf{x})}{p_g(\mathbf{x}) + p(\mathbf{x})} \tag{4.7}$$

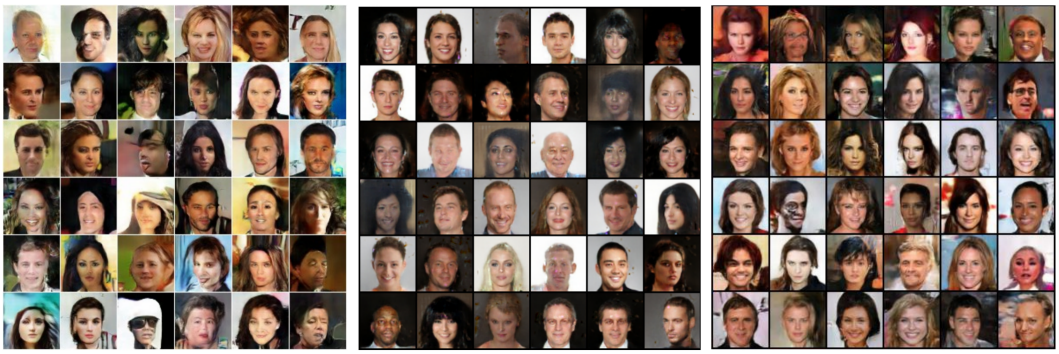
Under the assumption of infinite capacity of the discriminator, this equilibrium is reached if and only if $\forall \mathbf{x}\; p_g(\mathbf{x}) = p(\mathbf{x})$. We desire such an equilibrium, as at this point our generator is equivalent to the true data-generating function.

GANs are notoriously difficult to train. This owes to the inherent instability of the minimax game, whose optimum lies on a saddle point of the loss surface [Guttenberg (2016)]. In the three years immediately following 2014, GANs have become a particularly active sub-field of machine learning research. A number of interesting and creative applications have emerged. One consistent thread of GAN research is focused on training stability and the realism and resolution of the generated images. This thread remains relevant in the main SSL application of GANs we investigate in this thesis. In terms of image realism and resolution, as at time of writing *energy-based GANs* (EBGANs) are among the most stable and effective algorithms [Zhao et al. (2016)]. These approaches use an autoencoder as their discriminator, which attempts to maximise the difference in the distribution of reconstruction losses between the real and fake images. Figure 4 illustrates this current state-of-the-art in image generation with GANs.

Wasserstein GANs [Arjovsky et al. (2017)], Improved Wasserstein GANs [Gulrajani et al. (2017)] and DRAGAN [Kodali et al. (2017)] are the latest developments in a promising alternate direction for improving the stability of GAN training, which centre around replacing the implicit KL divergence minimisation that occurs in standard GAN training with an Earth-Mover (or Wasserstein-1) distance [Arjovsky et al. (2017)]. These approaches are also competitive in terms of stability and subjective generated image quality.

4.3.2 Approaches by which generative adversarial networks can be used for semi-supervised learning

At a high level, GANs can facilitate SSL as they are a means of extracting structure from unlabeled data. This structure can take many forms however, and can be used in a number of different ways. In our review of the literature, we have identified four overarching themes



(a) (b) (c)

Figure 4: Randomly generated celebrity faces from: (a) Energy-Based GAN [Zhao et al. (2016)], (b) Boundary Equilibrium GAN [Berthelot et al. (2017)] and (c) MAGAN [Wang et al. (2017)] (image from [Wang et al. (2017)].

in terms of how GANs can be used for SSL. In this section we detail several GAN-based SSL methods, and place each of these methods into one of our four broad categories.

Approach 1: Re-use the features from the discriminator

The first broad approach that can be found in the literature is to train a GAN by the original or some modified training procedure and loss, and then re-use the features from the discriminator in some way when training the classifier. This approach was used in Radford et al. (2015), one of the earliest attempts at performing SSL with GANs. There the authors train a GAN on the Imagenet-1k dataset and then re-use the learned features of the discriminator to train an SVM classifier on a labeled subset of CIFAR-10. Their results were competitive at the time, and particularly notable due to the fact that the features used by the classifier were learned on an entirely different dataset.

In general though, this broad approach of re-using discriminator features appears to have faded in popularity. This is likely due to the fact that, as we have discussed, SSL approaches that involve an unsupervised pre-training step, followed by a supervised training step, generally do not perform as well as more 'end-to-end' type approaches. It is likely that this general result also carries over to GAN-based SSL approaches.

That said, the method applied in Denton et al. (2016) would also fall into this category, and is quite effective. There the authors train a GAN where the generator is given images with missing patches and learns to in-paint the missing section. The discriminator then learns to discriminate between real images and these in-painted images. The penultimate layer of features of the discriminator is then shared with a classifier, whose cross-entropy loss is used in combination with the discriminator loss to learn the features. The authors assert that this particular task should lead to the discriminator learning semantic features about the images that are also useful for the classification task. Competitive results suggest that this is indeed the case, though as these are only presented for the STL-10 and Pascal VOC datasets, they are difficult to compare to most other methods discussed in this thesis.

It is likely that the 'all in one' approach Denton et al. (2016) adopt also contributes to their method's success; were the discriminator features instead pre-trained, then performance would likely drop. This method can also be viewed as training a classifier that is regularised by the GAN training task, and as such could also be categorised into the next broad approach we describe.

Approach 2: Use GAN-generated samples to regularise a classifier

A second approach is to use the generated samples of the GAN to regularise the classifier.

There are a number of different ways to achieve this. One example might be, if we assume we have a perfect generator, then we could encourage our classifier to minimise the entropy of its output on samples from that generator. This would then be equivalent to entropy minimisation on the unlabeled data per Grandvalet and Bengio (2005), but with an infinite amount of unlabeled data available. Other approaches [Springenberg (2015), Salimans et al. (2016), Dai et al. (2017)] instead (either implicitly or explicitly) *maximise* entropy on generated samples. This can also regularise a classifier, provided the generated samples are not 'too good.' These approaches are the focus of this thesis and are discussed in much further detail in the next section.

Approach 3: Learn an inference model during the GAN training process

A third approach involves training a GAN that also learns an inference model. Usually this means that an 'encoder' $E(\mathbf{x})$ is learned that transforms a real (or generated) sample back to the latent ($\mathbf{z}$) space of the GAN. Often these types of models encourage (implicitly or explicitly) the reconstruction error in the $\mathbf{z}$ space (i.e. $||\mathbf{z} - E(G(\mathbf{z}))||_2$) and/or in the $\mathbf{x}$ space to be low.

One example of this is Adversarially Learned Inference (ALI) [Dumoulin et al. (2016)], whose approach is identical to the concurrently-developed *Bidirectional GAN* [Donahue et al. (2016)]. In ALI, the discriminator is shown pairs of $\mathbf{x}$ and $\mathbf{z}$. It must output a high value for pairs of $\{\mathbf{x}_{real}, \mathbf{z}_{real}\}$ where $\mathbf{z}_{real} = E(\mathbf{x}_{real})$ and $\mathbf{x}_{real} \sim p(\mathbf{x})$, and low values for pairs of $\{\mathbf{x}_{fake}, \mathbf{z}_{fake}\}$, where $\mathbf{x}_{fake} = G(\mathbf{z}_{fake})$ and $\mathbf{z}_{fake} \sim p(\mathbf{z})$. As per the usual GAN set-up, the generator then tries to maximise the discriminator output for its fake pairs.

In Donahue et al. (2016) it is shown that in optimality, the encoder and generator in ALI learn inverse operations of one another. In Dumoulin et al. (2016) the authors use the encoder for SSL. The final three hidden layer activations of the encoder are concatenated with its output $\mathbf{z} = E(\mathbf{x})$. These 8960-dimensional feature vectors obtained for the labeled subset of the data set, and used to train an L2-SVM classifier. Through this method, the authors achieve competitive SSL results on the SVHN and CIFAR10 datasets (Table 4.2.1).

Some possible extensions to the ALI's approach could be suggested. First, ALI uses unsupervised pre-training followed by supervised learning on the extracted features. A more holistic approach might be more successful, for instance if the encoder were trained such that it directly received a categorical cross-entropy loss signal from the labeled data points. Ulyanov et al. (2017) introduce an alternative approach that allows the encoder to be estimated in a simpler, more efficient manner by training purely on the $\mathbf{z}$ reconstruction signal, $||\mathbf{z} - E(G(\mathbf{z}))||_2$. They coin their model *Adversarial Generator-Encoder networks*.

Approach 4: Use samples produced by a GAN as additional training data

Another approach that perhaps seems obvious is to use data produced by the generator of a GAN as additional training data. This requires a conditional generator, as we must have a target label for the new training data we produce. One key problem with directly applying this approach in SSL however is that typically we only have a few labels, and thus training a conditional generator that has both a diverse distribution and that still produces semantically-correct images according to the desired class label is presumably problematic.

Triple GAN [Li et al. (2017)] is one approach that gets around this problem to some extent, and as a result is able to use generated data as additional training data. In Triple GAN, a conditional generator $p_G(\mathbf{x}|y)$ is trained, in addition to a classifier $C(y|\mathbf{x})$, *and* a discriminator, which looks at pairs of $\mathbf{x}$ and y. After a sufficient number of training epochs has passed to ensure the generator's samples are accurate, generated samples are shown to the classifier, and the labels used to generate them are considered the targets. The cross-entropy loss on these samples is then added to the classifier's loss. This loss is given a smaller weight than the cross-entropy loss on real labeled samples. The results of this approach are competitive with other SSL methods from the same era (Table 4.2.1).

Kumar et al. (2017) also use generated samples as additional training data, albeit in a more implicit manner. This approach is an extension to Rifai et al. (2011) enabled by the

advent of GANs. In both of these approaches, it is assumed that the data lie on some lower-dimensional manifold. The central idea is then to have some way of estimating the *tangent space* to this manifold. By estimating this, we can regularise our classifier by encouraging it to be invariant to changes in our input that are in a tangential direction to the manifold. Put simply, if we take a data point, transform it to its manifold representation, move in a tangential direction to the manifold and then transform it back, our classifier output should not change, as the image should not have changed in any semantically meaningful way. In Rifai et al. (2011) the tangent space to the manifold is estimated via a contractive autoencoder, while in Kumar et al. (2017) the authors replace this with a pre-trained ALI model as described above. This approach also achieves competitive results (Table 4.2.1).

5 Model focus: Good Semi-Supervised Learning that Requires a Bad GAN

The model introduced in a paper titled *Good Semi-Supervised Learning that Requires a Bad GAN* (hereafter, BadGAN) [Dai et al. (2017)] forms the central focus of this thesis. This approach achieves the best SSL performance on PI-MNIST-100 and SVHN-1k of our reviewed methods, and is competitive on CIFAR10-4k (Table 4.2.1).

There are two closely related approaches to BadGAN that we first should introduce. These are from papers titled *Unsupervised and Semi-Supervised Learning with Categorical Generative Adversarial Networks* [Springenberg (2015)] and *Improved Techniques for Training GANs* [Salimans et al. (2016)]. We shall hereafter refer to the models introduced in these papers as *CatGAN* and *Improved GAN*, respectively. Before presenting these models however, we proceed to give a primer on prerequisite concepts surrounding Shannon's entropy and its relation to a classifier's decision boundaries.

5.1 Shannon's entropy and its relation to decision boundaries

Shannon's entropy $H\left(p(\cdot)\right)$ (hereafter 'entropy') is a functional that takes a probability density function $p(\cdot)$ as its argument and returns the *expected information* of a random variable x distributed according to that density function (i.e., $x \sim p(x)$). Intuitively, entropy reflects the *uncertainty* of a probability distribution (Figure 5.1).

Entropy:

$$\begin{aligned} H\left(p(\cdot)\right) &= \mathbb{E}[I(x \sim p(x))] \\ &= \mathbb{E}_{p(x)}[-\log p(x)] \\ &= \int -p(x) \log p(x) dx \\ &= -\sum_{k=1}^{K} p(k) \log p(k) \end{aligned}$$

Where the last line is the case when $p(\cdot)$ is a distribution over K classes.

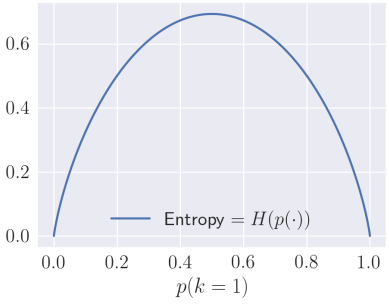


Figure 5: The entropy of a probability distribution over two classes $p(\cdot)$ as the probability of the first class changes from 0 to 1. Entropy is maximised when the probability of the two classes is equal.

We wish to demonstrate that higher entropy indicates the presence of a decision boundary. We focus here on the binary classification case. Let us imagine a linear two-class classifier that takes a two-dimensional input vector $\mathbf{x}$ as input and outputs a probability of the first class. Say it does this by taking that dot product of the input $\mathbf{x}$ with a vector of two weights $\mathbf{w}$, and then passing the scalar output through the logistic function. That is, $p(k = 1|\mathbf{x}, \mathbf{w}) = (1 + e^{-\mathbf{x}\cdot\mathbf{w}})^{-1}$. This is binary logistic regression.

We can see that the entropy-maximising input to this function is that where $\mathbf{x} \cdot \mathbf{w} = 0$. We can also see that, if there are points where $\mathbf{x} \cdot \mathbf{w}$ is positive, and points where it is negative, and our input space is continuous, then the region where $\mathbf{x} \cdot \mathbf{w} = 0$ will form a 'crossover point' where the most probable class of the output distribution changes. This region shall form this model's *decision boundary* (Figure 6). Hence, the decision boundary of a binary classifier is defined by the set of input points where the entropy of the output distribution is maximised.

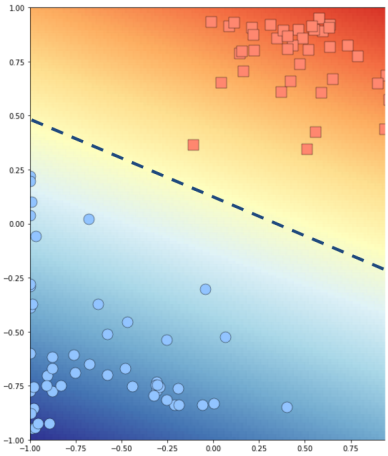


Figure 6: The decision boundary (dashed line) of a two-class linear classifier operating on a two-dimensional input space. At all points along the decision boundary, entropy is maximised.

To some extent, we expect the notion of decision boundaries being indicated by higher entropy to also extend to cases where we have more than two classes. It is difficult to prove this analytically, however, as when we have $K > 2$ classes, entropy is a function which maps a probability distribution in $\mathbb{R}^{K-1}$ to $\mathbb{R}^1$ space. If we hold all but two of our K classes constant, then it can be easily shown that the relationship between higher entropy and the decision boundary between those two classes holds. But without this assumption, there are regions of the input space where entropy can increase while we actually move *further* away from the decision boundary between two classes of interest. Intuitively though, the likelihood of being in such a region should decrease as the proportion of probability mass taken up by the two classes of interest increases. This is because, as the influence of the remaining classes on the entropy score decreases (as their combined probability mass approaches zero), we get closer to the situation where they can be considered constant, as their effect on the output of the entropy function becomes negligible.

5.2 CatGAN

CatGAN provides an approach for training a classification model and GAN simultaneously. We start with a regular GAN set-up, but modify the discriminator such that it has K outputs instead of just 1, where K is the number of class labels. These outputs are sent through a softmax activation and can thus be used to represent class probabilities for a given input. We then train via a minimax game similar to that of GANs, however rather than using the discriminator's direct output as a real/fake indicator, the entropy of the softmax of its K outputs now performs this role. We aim to optimise the following equation:

$$\max_G \min_D \mathcal{L}_{\text{xent}} + \mathcal{L}_{\text{ent}_U} - \mathcal{L}_{\text{ent}_G} \tag{5.1}$$

where:

$$\mathcal{L}_{\text{ent}_U} = \mathbb{E}_{\mathbf{x}\sim p_U}\left[H(C(\cdot|\mathbf{x}))\right] \tag{5.2}$$

$$\mathcal{L}_{\text{ent}_G} = \mathbb{E}_{\mathbf{x}_G\sim p_G}\left[H(C(\cdot|\mathbf{x}_G))\right] \tag{5.3}$$

Here $H(q(\cdot|c))$ is the conditional entropy of some distribution q given some c which q depends upon. As the discriminator output is now the same as our standard classifier's output (like that presented in Equation 4.1), we now use C to denote the discriminator function.

Through its version of the GAN minimax game, CatGAN is capable of learning to mimic the true data distribution. As discussed in the previous section, entropy reflects the 'spikiness' of the output distribution for a given input. When the entropy of the discriminator output $H(C(\cdot|\mathbf{x}))$ is minimised, all of its probability mass is in a single class. Through this mechanism, the $\mathcal{L}_{\text{ent}_U}$ term encourages the discriminator to make confident ('spiky') predictions on unlabeled data points. Similarly, the $-\mathcal{L}_{\text{ent}_G}$ term encourages the discriminator to output a flat class distribution for generated samples.

Meanwhile, the generator wishes to produce samples that the discriminator makes confident ('spiky', or low entropy) predictions on. Thus, in the absence of the cross-entropy loss term on the labeled data, and with unlimited capacity, the discriminator should be able to learn an equivalent function to that of the original GAN formulation. Thus in theory, the generated images should be able to reach the same quality as in a regular GAN. In practice however, we find that the quality of the samples produced by CatGAN is lower, presumably due to the more complex parameterisation of this form of GAN.

CatGAN can be used for SSL by including a categorical cross-entropy loss term $\mathcal{L}_{\text{xent}}$ (the authors show that without this term, it can also be used effectively for K-means clustering). CatGAN achieves competitive SSL results on PI-MNIST-100 and CIFAR10-4k (Table 4.2.1). The authors however, do not present any clear mechanism as to how the addition of a generator enables CatGAN to achieve these SSL results. We elucidate the mechanism by which this is likely occurring later in this section (Section 5.4).

Like in the original GAN set-up, we expect the CatGAN minimax game to ultimately reach an equilibrium, where neither the generator nor discriminator can improve further. This equilibrium could occur in two places. The first is where the generator has learned to match the true data distribution. At this point, the discriminator cannot distinguish between real and generated samples. In the semi-supervised setting, model performance is likely to suffer if this point is reached, as if $G(p_z(\mathbf{z})) = p_U(\mathbf{x})$, then the two entropy penalty terms in Equation 5.1 will cancel one another out, and we are left with just the supervised cross-entropy loss. As we discuss in Section 5.4, an equivalent result for the Improved GAN model is derived in Dai et al. (2017).

The other equilibrium is that where the generator reaches a point where it is producing samples off the data manifold, but that the discriminator still makes confident predictions on. This could only occur if there are outliers in the dataset, where there are 'real' data points not on the true data manifold, or if the discriminator does not have the capacity to separate out some region that is not on the data manifold from other regions that are on the manifold, and thus makes high-confidence predictions for both these on- and off-manifold regions. It seems that both of these scenarios are unlikely, and thus we generally expect the CatGAN model to converge to the first of these described equilibria.[4]

5.3 Improved GAN

A number of novel techniques designed to address the performance and stability of GANs are introduced in Salimans et al. (2016). We briefly cover these techniques here, and then proceed to describe their specific approach to SSL with GANs.

One important technique introduced is *feature matching*. Here, the hidden layer activations from a layer of the discriminator (usually the average pooling layer following the convolutional layers in a discriminator), are averaged over each unlabeled batch, and over each generated batch. This gives two vectors of average features, one from the real data and one from the fake data. The L2 or L1 norm of the difference between these vectors is then added as a cost to the generator. Through this, the generator gains an additional training signal that encourages it to produce images whose features, according to the discriminator, match those of real images.

[4]Thank you to Zihang Dai for useful input regarding potential equilibria.

Other techniques introduced include *minibatch discrimination*, which encourages within-batch diversity of the generated images, and *historical parameter averaging*, which penalises model parameters for straying too far from their historical running averages to improve training stability. *One-sided label smoothing* essentially replaces the 'label' of 1 for the discriminator's target value for real data with a softer target of 0.9 to improve numerical stability. Finally, with *virtual batch normalisation* the authors avoid issues related to using batch normalisation [Ioffe and Szegedy (2015)] with GANs by using a fixed batch of real data to calculate the normalisation constants at each layer of the discriminator, rather than using whichever batch is currently being fed through.

5.3.1 The Improved GAN SSL Model

What we call *Improved GAN* refers to the model comprised by the specific combination of these techniques that Salimans et al. (2016) use to achieve their reported SSL results. To combine GAN training with supervised learning, Improved GAN, like CatGAN increases the number of discriminator outputs. Improved GAN's discriminator has $K+1$ outputs instead of just K, however. These $K+1$ outputs are passed through a softmax to obtain a probability distribution (i.e., the $K+1$ outputs sum to 1 after the softmax). The $K+1$th probability is then used to reflect the probability, according to the discriminator, that a given input data point is from the true data distribution.

The authors note that the existence of the $K+1$th vector of weights linking the penultimate layer of the discriminator to the softmax output layer is unnecessary; with this weights vector, the model is over-parameterised (due to the fact that the softmax always sums to 1). As such, this vector of weights is fixed to zero. After this change, it is shown that $D(\mathbf{x})$ – the probability that a given $\mathbf{x}$ is from the real data distribution – reduces to $\frac{Z(\mathbf{x})}{Z(\mathbf{x})+1}$, where $Z(\mathbf{x}) = \sum_{k=1}^{K} \exp\left(\mathbf{w}_k^T \cdot \mathbf{x}\right)$. This means in practice our discriminator only needs to have K outputs. The discriminator then uses the magnitude of the K class logits relative to 1 to discriminate between real and fake data.

We can now add a cross-entropy loss to our standard GAN minimax game (as originally defined in Equation 4.6):

$$\max_G \min_D \mathcal{L}_{\text{xent}} - \mathbb{E}_{\mathbf{x}\sim p_U(\mathbf{x})}\left[\log(D(\mathbf{x}))\right] - \mathbb{E}_{\mathbf{z}\sim p_z(\mathbf{z})}\left[\log(1 - D(G(\mathbf{z})))\right] \tag{5.4}$$

here the $C(\cdot|\mathbf{x})$ function required by $\mathcal{L}_{\text{xent}}$ (as it was first defined in Box 2) is defined as the softmax over the first K of the discriminator's $K+1$ outputs.

However, the authors find that this particular minimax game does not perform so well for the purposes of SSL. Instead, they find a much more effective approach is to minimise the above loss for the discriminator only. Meanwhile, the generator is trained using only feature matching. This gives us the following loss function for the discriminator:

$$\mathcal{L}_{\text{dis}} = \mathcal{L}_{\text{xent}} - \mathbb{E}_{\mathbf{x}\sim p_U(\mathbf{x})}\left[\log(D(\mathbf{x}))\right] - \mathbb{E}_{\mathbf{z}\sim p_z(\mathbf{z})}\left[\log(1 - D(G(\mathbf{z})))\right] \tag{5.5}$$

and for the generator:

$$\begin{aligned} \mathcal{L}_{\text{gen}} &= \left|\left| \mathbb{E}_{\mathbf{x}\sim p_U(\mathbf{x})}\left[f(\mathbf{x})\right] - \mathbb{E}_{\mathbf{z}\sim p_z(\mathbf{z})}\left[f(G(\mathbf{z}))\right] \right|\right|^2 \\ &= \mathcal{L}_{\text{FM}} \end{aligned} \tag{5.6}$$

where $f(\cdot)$ is a readout of the activations of some intermediate layer of the discriminator (usually the penultimate layer of activations before the output layer).

Salimans et al. (2016) note that, despite producing subjectively lower-quality images, this feature matching approach produces far better SSL results (Table 4.2.1) than approaches that subjectively produce much better-looking images, such as one using minibatch discrimination. When we discuss BadGAN in the next section a theoretical argument is presented as to why this is the case.

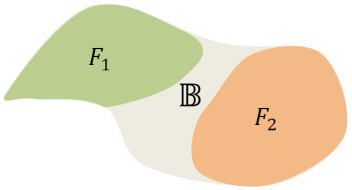


Figure 7: Simple illustration of the hypothetical bound $\mathbb{B}$, which is a region containing the union of all F_k's. Here an ideal complement generator will produce points in the region between the F_1 and F_2.

5.4 BadGAN

Both CatGAN and Improved GAN were able to achieve competitive SSL results across several benchmark datasets (Table 4.2.1). Curiously though, in both these papers the images generated as a by-product of training the semi-supervised model do not appear to be of particularly good subjective quality. In Dai et al. (2017), a likely explanation for this phenomenon is given, and this prompts the introduction of the BadGAN model.

The starting point in Dai et al. (2017) is the Improved GAN model used for SSL as described in the previous section. The authors then consider the situation of a perfect generator, that is $\forall \mathbf{x}\ p_G(\mathbf{x}) = p_U(\mathbf{x})$ (where p_G is the probability distribution defined by sampling $\mathbf{z} \sim p_{\mathbf{z}}(\mathbf{z})$ and placing it through the generator $G(\mathbf{z})$). The authors provide a proof that when the generator is perfect, the learned classifier shall be no better than a supervised-only classifier trained on just the labeled data points. In Section 5.2, we showed that this is also likely the case for CatGAN under the assumption of a perfect generator.

Having pointed out this problematic aspect of Improved GAN, the authors in Dai et al. (2017) then proceed to define what they call a *complement generator* as follows. Let f be a function that maps data points in the input space to the feature space (feature space here meaning the hidden layer activations of the penultimate layer in a classifier). Given thresholds $\{\epsilon_k\}_{k=1}^K$, the authors define the support of each class in this feature space to be $F_k = \{f(\mathbf{x}) : p(\mathbf{x}|\text{class}(\mathbf{x}) = k) > \epsilon_k\}$. In other words, F_k is the union of all points in the feature space where input points $\mathbf{x}$ mapped to the feature space could possibly fall, if their class is k and their probability $p(\mathbf{x})$ is greater than ϵ_k. The F_k's are assumed to be disjoint sets, which can always be made true by adjusting the thresholds, so long as the F_k's do not share any mode.

The union of all F_k's is then assumed to be bounded by a convex set $\mathbb{B}$ (see Figure 7 for an illustration). A complement generator G_C, defining a distribution over $\mathbf{x}$, $p_{G_C}(\mathbf{x})$ is then defined as one whose support in the feature space F_{G_C}, is a relative complement set in $\mathbb{B}$, that is $F_{G_C} = \mathbb{B} - \cup_{k=1}^K F_k$.

The authors then go on to show that, given these assumptions, if $\mathcal{L}_{\text{dis}}$ in Equation 5.5 is minimised on a finite training set of labeled, unlabeled and p_{G_C}-generated samples, the discriminator learns a correct decision boundary on all training data points. That is, "correct decision boundaries are learned on each high-density subset F_k (defined by ϵ_k) of the data support in the feature space" [Dai et al. (2017)].

With this theoretical result on hand, and after showing a proof of concept on synthetic examples, the authors turn to defining a framework that might be capable of learning such a complement generator. They define a target distribution for the desired complement generator p^* as follows:

$$p^*(\mathbf{x}) = \begin{cases} \frac{1}{Z}\frac{1}{p(\mathbf{x})} & \text{if } p(\mathbf{x}) > \epsilon \text{ and } \mathbf{x} \in \mathbb{B}_x \\ c & \text{if } p(\mathbf{x}) \leq \epsilon \text{ and } \mathbf{x} \in \mathbb{B}_x \\ 0 & \text{otherwise} \end{cases}$$

where c is some small constant. This definition states that the complement generator should produce points with low probability, but that still fall in the feature space bound $\mathbb{B}$.

The authors then derive the KL divergence between p^* and the distribution defined by the generator p_G:

$$D\left(p_G \| p*\right) = -H(p_G) + \mathbb{E}_{\mathbf{x}\sim p_G} \log p^*(\mathbf{x}) \delta\left[p(\mathbf{x}) > \epsilon\right] \tag{5.7}$$

This also needs to be constrained such that p^* only produces points in $\mathbb{B}_x$. To enforce this, the authors simply add the feature matching cost from Improved GAN, as described in Section 5.3. This gives the following minimisation problem for the generator:

$$\min_G \; -H(p_G) + \mathbb{E}_{\mathbf{x}\sim p_G} \log p^*(\mathbf{x}) \delta\left[p(\mathbf{x}) > \epsilon\right] + \mathcal{L}_{\text{FM}} \tag{5.8}$$

To optimise 5.8, the authors needed to define proxies for $H(p_G)$ and $p(\mathbf{x})$ (as the expectation can be approximated by Monte Carlo sampling from p_G). Two approximation alternatives for $H(p_G)$ are given. The first is the *pull-away term* originally proposed in Zhao et al. (2016), which adds a loss term $\mathcal{L}_{\text{PT}} = \frac{1}{N(N-1)} \sum_{i=1}^{N} \sum_{j\neq i} \left(\frac{f(\mathbf{x}_i)^T f(\mathbf{x}_j)}{||f(x_i)||_2^2 \, ||f(x_j)||_2^2} \right)$ that encourages the generated feature vectors to be orthogonal. This encourages diversity, which can be thought of as a proxy for entropy [Dai et al. (2017)].

The other suggested proxy for entropy introduces an additional encoder q and defines a variational upper bound $-H(p_G) \leq -\mathbb{E}_{\mathbf{x},\mathbf{z}\sim p_G}\left[q(\mathbf{z}|\mathbf{x})\right]$, where $q(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mu(\mathbf{x}), \sigma^2(\mathbf{x})$, and $\mu(\cdot)$ and $\sigma^2(\cdot)$ are neural networks. Making use of the reparameterisation trick introduced in Kingma and Welling (2013), this upper bound can then be minimised as part of Equation 5.8.

Also required by Equation 5.8 is some way of estimating the density $p(\mathbf{x})$ for any given $\mathbf{x}$. The authors' choice is to use a PixelCNN++ model [Salimans et al. (2017)], pre-trained on the training dataset. PixelCNN++ is a large model, taking around 5 days to converge when training on 8 powerful GPUs. Due to this, one of our central research questions involves investigating alternative density measures to potentially replace this component of the BadGAN model.

With this redefined version of the generator loss of the Improved GAN model on hand, the authors proceed to make one small change to its discriminator loss. They note that there is no term in Equation 5.5 that enforces the model to be confident in its classification decisions on unlabeled data points. To encourage this, the entropy of the unlabeled data points (Equation 5.2) is added. Thus the discriminator (or classifier) loss becomes:

$$\mathcal{L}_{\text{dis}} = \mathcal{L}_{\text{xent}} + \mathcal{L}_{\text{ent}_U} - \mathbb{E}_{\mathbf{x}\sim p_U(\mathbf{x})}\left[\log(D(\mathbf{x}))\right] - \mathbb{E}_{\mathbf{z}\sim p_z(\mathbf{z})}\left[\log(1 - D(G(\mathbf{z})))\right] \tag{5.9}$$

With the generator and discriminator loss functions redefined, the authors then proceed to fit the BadGAN model to the standard benchmarks, achieving results setting the current state-of-the-art (for leaner classifier architectures) on all three benchmarks (Table 4.2.1).

5.4.1 Implications of the BadGAN model

The findings in Dai et al. (2017) which led to the BadGAN model provide an explanation for why both the CatGAN and Improved GAN models appear to produce better SSL results with subjectively worse images. The paper shows that the Improved GAN model formulation actually lends itself better to a complement generator. By producing images that are of low probability, but that still fall into the feature space of the classifier, the generated images have the effect of regularising the classifier in Improved GAN.

In the next section, we introduce a number of regularisers commonly found in SSL literature and analyse their performance both theoretically and through synthetic experiments. We find that an analysis of these regularisers and their shortcomings motivates a need for a

model like BadGAN. We then also evaluate this type of model theoretically and on synthetic data.

Additionally, in Section 6.3, we give more concrete theoretical evidence that, despite its different loss function, the CatGAN model likely performs SSL through essentially the same mechanism as Improved GAN. This suggests that the implications of BadGAN for the Improved GAN model should also carry over to CatGAN.

Part III
Analysis and experiments

6 Enforcing low-density separation

We now return to one of our original SSL assumptions. The low-density separation assumption (Assumption 3.1.3) asserts that in some representation of our data (be it the original space or some manifold of that space), most of our data will lie in high-density regions, and the classes will be separated by regions of low-density. As mentioned in Section 3.1.3, although this assumption is equivalent to the cluster assumption (Assumption 3.1.2), it can inspire different algorithms and approaches to SSL.

In the first part of this section, based on the low-density separation assumption, we motivate a range of loss function components that are commonly used across SSL literature. We illustrate the behaviour of each component by analysing its performance on two synthetic datasets. The analysis of these different loss function components naturally leads to a desire for a BadGAN-like model, which we then also introduce and evaluate.

As all of these loss functions are based on entropy in some way, we follow this with a more theoretical investigation of different proxies for entropy, and also draw connections between the CatGAN and Improved GAN approaches. Based on this analysis we make some claims as to which of these approaches might work best in practice.

The analyses in this section give a foundation for understanding the various loss component options we have for achieving low-density separation. This allows us to make hypotheses about which combinations of these components might be most effective in the context of SSL with neural networks. This then forms the basis for some of our research questions and hypotheses (Section 7), which we then address through larger empirical experiments (Section 8).

6.1 Approaches to low-density separation based on entropy

Here we introduce a number of popular ideas from the SSL literature. These ideas all continue along a similar thread of aiming to learn a classifier that places its decision boundaries in low-density regions of the data or manifold space through the use of entropy. This thread culminates in a desire for a model which can *generate* points in these low-density regions (i.e., a model like BadGAN).

6.1.1 Synthetic experiments used in this section

To illustrate the impact of the various cost functions introduced in this section, we conduct a series of synthetic experiments aimed at illustrating the behaviour and performance of a neural network trained with that cost function. The setup of these experiments is described in Box 3.

Box 3: Description of the setup for our synthetic experiments

We run simple experiments on two synthetic datasets to give an overview of the behaviour of the various cost functions presented in this section. Both datasets consist of a set of two-dimensional input vectors $\mathbf{X}$, with corresponding labels $\mathbf{y}$. Each dataset was split into 10,000 training examples, 10,000 validation examples, and 20,000 test examples. To make the experiments semi-supervised, most of the labels were then removed from the training set.

The *Gaussian Mixture* dataset was constructed by sampling from three Gaussian distributions. The left distribution (Figure 8, blue circles) was sampled from 50% of the time and labeled as class one. The two rightmost distributions (red squares) were sampled from the remaining 50% of the time and were labeled as class one. All but 25 labels per class were then removed from the training set. It was also ensured that the labels kept for the second class were only from the lower-right Gaussian distribution. This was done to illustrate the behaviour of the algorithms we introduce when faced with a cluster with no labels.

The *Moons* dataset was generated using a built-in method from the Python *sklearn* [Pedregosa et al. (2011)] package. Just 5 labels per class were left in the training set. This dataset is commonly used to illustrate the behaviour of SSL algorithms, as learning a correct classifier with only a few labels relies heavily on the low-density separation assumption.

In each of these synthetic experiments, we train a feedforward neural network with a two-dimensional input layer, two hidden layers with 64 units each and ReLU activations, one more hidden 'feature space' layer with two units and tanh activations, and finally a two-dimensional output layer with a softmax activation. The two-dimensional, tanh-activated feature space layer was included out of a desire to have a visualisable feature space that occupied a bounded interval. We regularised the model by adding a typical L2 regularisation term, $\mathcal{L}_{\mathrm{L}_2} = \frac{\lambda}{2}\sum_i ||\boldsymbol{\theta}_i||_2^2$, where $\boldsymbol{\theta}_i$ is the weights matrix of the ith layer of the neural network. Unless otherwise noted, λ was fixed at 0.001. Models were trained using Tensorflow [Abadi et al. (2015)].

A training step or iteration consisted of sampling 100 labeled examples (with replacement) and 100 unlabeled examples, feeding them through the network, and backpropagating the gradients of the cost function to update the weights. Supervised-only experiments with all labels available clearly showed that the network and training procedure were more than capable of learning a correct decision boundary for the two datasets and achieved above 99% accuracy on the test set. We train the models until convergence and then plot the learned $p(k|\mathbf{x})$ by showing the models output probability over the entire input space (again Figure 8, for example).

Looking at the results (Figure 8) from our synthetic experiment using just the cross entropy loss, we can see that due to the lack of labels, the supervised-only setup does not perform well on either dataset. It slices in half the upper-right Gaussian mixture component in the Gaussian Mixture dataset, and it fails to notice the intricacies of the Moons dataset. It certainly appears that in both these cases the model could improve substantially if it were encouraged to place its decision boundaries in low-density regions of the input space. To this end, we proceed to formulate an additional term for our loss function.

6.1.2 Using entropy to incorporate the low-density separation assumption into our model

The low-density separation assumption states that "the decision boundaries [for a classification model] should lie in a low-density region" of the data or manifold space [Chapelle et al. (2006)]. As we showed in Section 5.1, the decision boundaries of a model should correlate

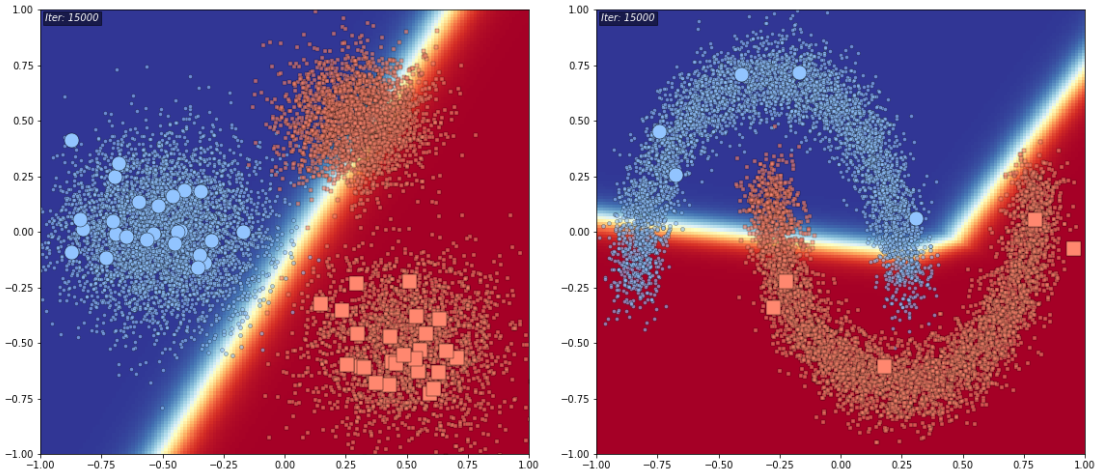


Figure 8: Classification and decision boundary results from a neural network trained in a supervised-only fashion (i.e., using only $\mathcal{L}_{\text{xent}} = \mathbb{E}_{\mathbf{x},y\sim p_L(\mathbf{x},y)}$) on the *Gaussian Mixture* (left) and *Moons* (right) datasets. These figures show the input data points to the neural network. Points from the first class are blue circles; from the second class, red squares. Large points indicate available training labels. The background colour indicates the decision regions learned by the model.

with regions where entropy is high. Given this, we can equivalently state the low-density separation assumption as saying that 'high entropy outputs should only be produced by input points of low probability.'

By definition however, we do not have many input points in our dataset that are of low probability. Without such points, it is difficult to determine where exactly to tell our model to output high entropy distributions. One workaround to this problem is to instead enforce the inverse of our assumption, which can be stated as 'high probability points should result in low entropy outputs.' Put simply, this means that the model should make confident predictions on the data that we have (assuming that, by virtue of its existence, the data we have is of high probability). This particular interpretation has inspired a number of algorithms, all based on the idea that an optimal classifier should have low entropy in high-density regions. We motivate and introduce these algorithms here.

We are seeking a function that both classifies the labeled data correctly, and minimises the entropy of its output probability distribution in high-density regions. One way we can penalise high entropy outputs in high density regions is through a loss function component where we multiply the entropy of each point by the density at that point:

$$\begin{aligned}\mathcal{L}_{\text{ent}} &= \int H(p(\cdot|\mathbf{x}))p(\mathbf{x})d\mathbf{x} \\ &\approx \mathbb{E}_{\mathbf{x}\sim p_U(\mathbf{x})}\left[H(C(\cdot|\mathbf{x}))\right] \\ &= \mathcal{L}_{\text{ent}_U}\end{aligned} \tag{6.1}$$

Here we have made the commonly-used assumption that the density of our unlabeled dataset approximates the true density of $\mathbf{x}$. Weighting this term by γ, we can combine it with the cross-entropy loss on the labeled data points to obtain:

$$\mathcal{L}_{\text{entmin}}(\boldsymbol{\theta};\mathbf{X},\mathbf{y},\gamma) = \mathcal{L}_{\text{xent}} + \gamma\mathcal{L}_{\text{ent}_U} \tag{6.2}$$

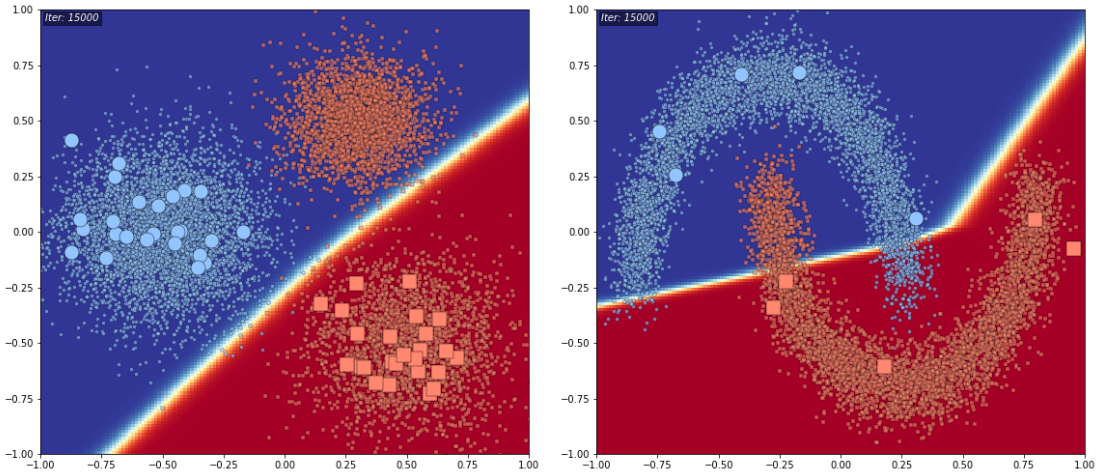


Figure 9: The result of training our neural network on the *Gaussian Mixture* and *Moons* datasets using $\mathcal{L}_{\text{entmin}}$ (Equation 6.2) as our loss function for 15,000 training steps.

This is exactly the loss function introduced in Grandvalet and Bengio (2005).

As mentioned in Box 2, we can approximate the expectations in our loss function through Monte Carlo sampling, and update our neural network via stochastic gradient descent. Figure 9 shows the converged solution of our model trained with this loss function on the synthetic datasets. Unfortunately it appears that the entropy minimisation term has not resolved the issues exhibited by the supervised-only run (Figure 8). Compared to the supervised-only run, the decision boundaries appear to be thinner – that is, the model changes its most probable class more rapidly. This is expected, as a steeper decision boundary means that the high-entropy region will be smaller, which in turn means there will be less chance for points to fall in that region, and therefore less of a penalty for not minimising entropy for those points.

This behaviour illustrates the shortcoming of entropy minimisation in isolation; if the decision boundary is in the middle of a high-density region, and the density has a flat or noisy gradient in all directions, then there will be no strong force pulling the decision boundary toward lower density regions. Hence in the Moons dataset, the decision boundary remains slicing through the lower-right moon, as density is relatively constant over the entire moon shape. In the Gaussian Mixture dataset on the other hand, density is reducing in all directions away from the centre of each mixture component, so the model has an incentive to move the decision boundary increasingly further from these centres, though not in any particular direction. Unfortunately in this case this has led to misclassifying the entire upper-right cluster.

6.1.3 Taking advantage of a known prior class distribution

Along a similar line of inquiry, in Krause et al. (2010) the authors note that entropy minimisation in isolation can lead to the degenerate solution of having no decision boundaries, with instead all data points being assigned to one class.[5] The proposed remedy is an additional loss function term that aims to maximise the entropy of the average class distribution (the main loss criterion put forward in that paper also suggests a regularisation term that is

[5]This concern is of particular significance in the unsupervised setting, but is still relevant here.

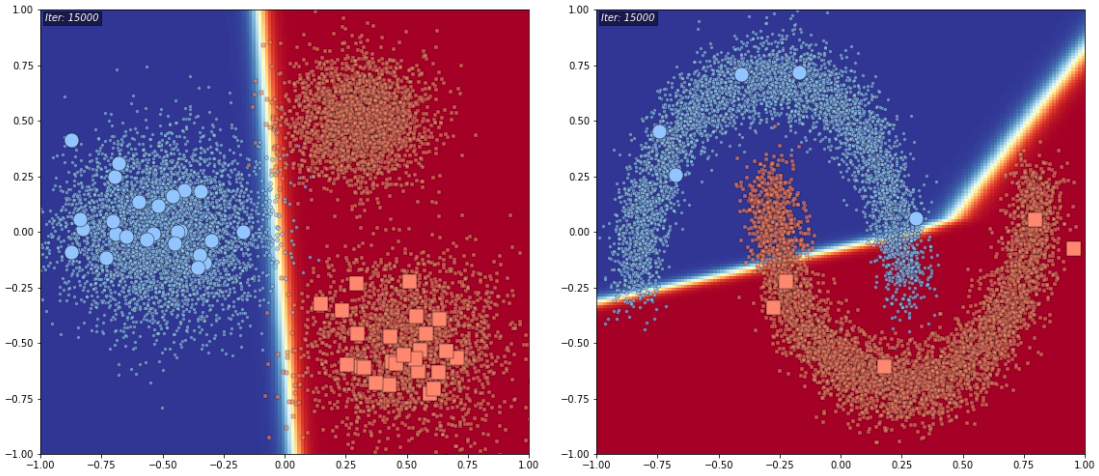


Figure 10: The result of training our neural network on the *Guassian Mixture* and *Moons* datasets using the RIM loss function described in Equation 6.3 for 15,000 training steps.

omitted in the equation here, but as explained in Box 3, is present in the form of L_2 regularisation in our neural network). Adding this additional term to $\mathcal{L}_{\text{entmin}}$ (Equation 6.2) gives us a regulariser that is equivalent to the empirical estimate of the mutual information between $\mathbf{X}$ and $\mathbf{y}$; $I_{\boldsymbol{\theta}}(\mathbf{y}; \mathbf{X})$ [Krause et al. (2010)]:

$$\mathcal{L}_{\text{RIM}} = \mathcal{L}_{\text{xent}} + \gamma \mathcal{L}_{\text{ent}_U} - \eta H\left(\bar{C}\right) \tag{6.3}$$

Where $\bar{C} := \mathbb{E}_{\mathbf{x} \sim p(\mathbf{x})}\left[C(\cdot|\mathbf{x})\right]$, the average probability distribution over all data points implied by the model.

This additional penalty term, $-\eta H\left(\bar{C}\right)$, has the effect of encouraging an equal number of points to be assigned to each class, which is desirable if we have the prior belief or knowledge that our dataset should contain an (approximately) equal number of points from each class. The beneficial impact of this is illustrated by this loss function's performance in our experiment (Figure 10). We can see that while performance on the Moons dataset is unchanged from the previous experiment, on the Gaussian Mixture dataset the new penalty has encouraged the model to put an equal number of points in each class.

We show in Section 6.2 that this particular penalty term implicitly assumes that the unknown labels are evenly distributed over the K classes. Since this implicit prior distribution is consistent with the data distribution, performance has improved, with the model now putting its decision boundary on the correct side of the unlabeled cluster. It should be noted that this assumption of a flat prior may not be appropriate in cases where the class distribution of the unlabeled dataset is strongly skewed. In those cases the KL divergence between a skewed prior derived from background knowledge about the data and the average distribution $\bar{p}$ may serve well as an alternative, though we leave experimenting with this for future work.

We should also note that for this penalty to work on our synthetic datasets we found that γ needed to be less than η (specifically we used $\gamma = 0.5$, $\eta = 1.0$). Otherwise, the entropy minimisation cost would penalise the model too heavily for putting its decision boundary through the upper-right cluster whilst converging towards the average-entropy-maximising state of including the entire cluster in the second class (red) decision region. This in turn would prevent the optimal decision function from being reached.

6.1.4 Generating data in low-density regions of the input space

In the previous section, we derived the idea for entropy minimisation based on the inversion of the principle that regions of low density (low $p(\mathbf{x})$) should result in high entropy outputs; as we did not have these low-density inputs, we instead asserted that regions of high density should have low-entropy outputs.

We now return to the original proposition, and instead assume that we can *generate* datapoints of low $p(\mathbf{x})$, and encourage our model to place its decision boundaries at these points. The setup we give here is very similar to that assumed in the BadGAN model (Section 5.4), except now our bound $\mathbb{B}$ applies to the input space, rather than the feature space, as in these lower-dimensional synthetic examples providing good coverage of the input space is not an issue. In the BadGAN paper [Dai et al. (2017)], it is shown that the same conclusions we make still hold when working in the feature space.

We assume our $p(\mathbf{x})$ is such that for some very small probability threshold ϵ, all $\mathbf{x}$ values where $p(\mathbf{x}) > \epsilon$ are bounded by some convex set $\mathbb{B}$ (in our synthetic experiments we just use the interval {-1, 1} in both dimensions for $\mathbb{B}$). We define another probability distribution p_G, that is also bounded in the same way by $\mathbb{B}$, but that generally produces points of low $p(\mathbf{x})$ as follows:

$$p_G(\mathbf{x}) = \begin{cases} \frac{1}{Z} & \text{if } p(\mathbf{x}) < \epsilon \text{ and } \mathbf{x} \in \mathbb{B} \\ 0 & \text{otherwise} \end{cases} \tag{6.4}$$

Here Z is a normalisation constant to ensure p_g integrates to 1, and ϵ is a threshold defining the maximum $p(\mathbf{x})$ we will allow our distribution p_g to generate points of.

We then add an additional term to the loss function in Equation 6.3 that encourages our model to have high entropy for these generated points. This term is constructed as follows:

$$\begin{aligned} \mathcal{L}_{\text{ent}_G} &:= \int H(C(\cdot|\mathbf{x}))p_G(\mathbf{x})d\mathbf{x} \\ &= \mathbb{E}_{\mathbf{x}_G \sim p_G(\mathbf{x}_G)} \left[H(C(\cdot|\mathbf{x}_G)) \right] \end{aligned} \tag{6.5}$$

Our complete loss function then becomes:

$$\mathcal{L} = \mathcal{L}_{\text{xent}} + \gamma \mathcal{L}_{\text{ent}_U} - \eta H\left(\bar{C}\right) - \beta \mathcal{L}_{\text{ent}_G} \tag{6.6}$$

$$= \mathcal{L}_{\text{RIM}} - \beta \mathcal{L}_{\text{ent}_G} \tag{6.7}$$

We tested this approach on our synthetic datasets by taking the training data and estimating $p(\mathbf{x})$ over the input space by means of a Gaussian kernel density estimate (KDE). After making this KDE, it was used to give an array of coordinates where $p(\mathbf{x}) < \epsilon$. We could then approximately sample from p_G as it was defined in Equation 6.4 by taking one of these points and adding a very small amount of Gaussian noise to it.

We set γ, η and β all to 1.0 and ran our synthetic experiments. The results (Figure 11) suggest that this approach, like InfoReg, also converges to a solution that favours placing decision boundaries in low-density regions. The uncertainty (indicated by the light-coloured regions) that results in regions of low density is also a desirable property; ideally we do not want our model to be very confident in regions where there is (almost) no data (though on the other hand, if learning a function that is uncertain in regions where $p(\mathbf{x})$ is effectively zero costs us modelling capacity in regions where we actually have data, then this is a waste of modelling capacity).

6.2 Viewing entropy maximisation as minimising a Kullback-Leibler divergence

In the previous section, we motivated a number of loss function components that can be added to a neural network classifier. These all rely upon minimising or maximising entropy. As such, in this and the following section we investigate alternative proxies for entropy, so

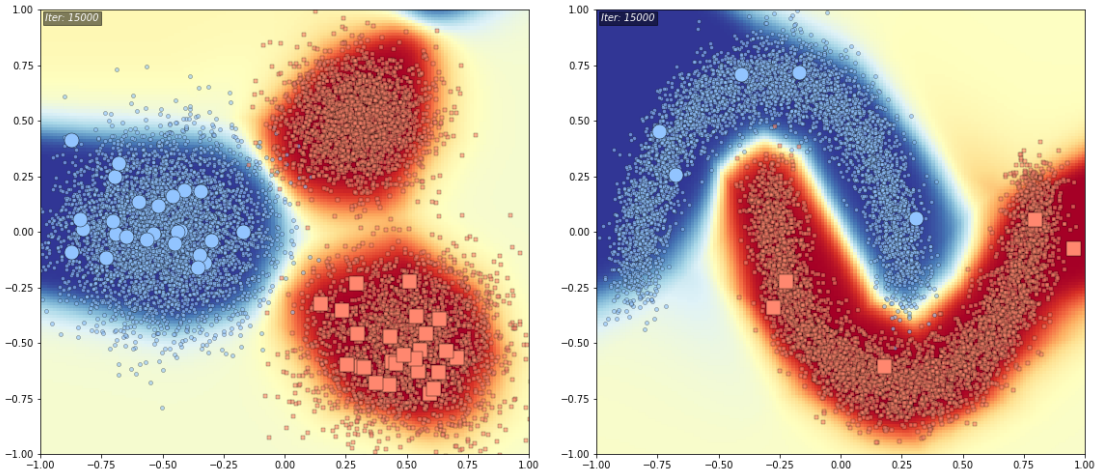


Figure 11: The result of training our neural network on the *Guassian Mixture* and *Moons* datasets with a low-probability data generator for 30,000 training steps.

as to assess whether Shannon entropy is indeed the most appropriate functional for our purpose. We find that insights and alternative loss functions can be gained by formulating our entropy minimisation or maximisation goals as KL divergences between the model's output probability distribution and certain target distributions.

In Equation 6.3, an entropy maximisation term $-H(\bar{C})$, where $\bar{C} := \mathbb{E}_{\mathbf{x}\sim p(\mathbf{x})}\left[C(\cdot|\mathbf{x})\right]$, was introduced to encourage the model to evenly distribute probability mass across classes (though at an aggregate, rather than individual sample level). We find that maximising entropy in this way implicitly assumes a prior distribution on the unlabeled data, insofar as it is equivalent to the KL divergence between the average model probability distribution $\bar{C}$ and that of a flat prior $q(\cdot) := \forall k : q(k) = 1/K$, plus a constant (omitting $\bar{C}$'s dependence on the data $\mathbf{X}$):

$$\begin{aligned} D_{KL}\left(\bar{C}\|q\right) &= \sum_{k=1}^{K} \bar{C}(k) \log \frac{\bar{C}(k)}{q(k)} \\ &= \sum_{k=1}^{K} \bar{C}(k) \log K\bar{C}(k) \\ &= \sum_{k=1}^{K} \bar{C}(k) \log \bar{C}(k) + \log K \\ &= -H(\bar{C}) + \log K \end{aligned} \tag{6.8}$$

We refer to Equation 6.8 as the 'original KL' loss (though since it is equivalent to maximising entropy, we also refer to it as just entropy, or sometimes the 'CatGAN approach,' as entropy is used by the CatGAN model's discriminator).

Another contribution we make here is to also reverse this KL divergence to devise an

alternative entropy-maximising penalty:

$$D_{KL}\left(q\middle\|\bar{C}\right) = -\sum_{k=1}^{K} q(k)\log\bar{C}(k) + \sum_{k=1}^{K} q(k)\log q(k)$$
$$= -\frac{1}{K}\sum_{k=1}^{K}\log\bar{C}(k) - \log K \tag{6.9}$$

We coin this expression the 'Reverse KL' loss. Like the original KL loss, the Reverse KL loss is minimised when $\forall k : \bar{C}(k) = q(k) = 1/K$. However, this cost function's response as the average distribution's entropy declines is much steeper (Figure 12). In the next section we theoretically consider the potential benefits and drawbacks of these two alternative formulations, and also draw connections between these alternatives and the Improved GAN versus CatGAN discriminator loss functions.

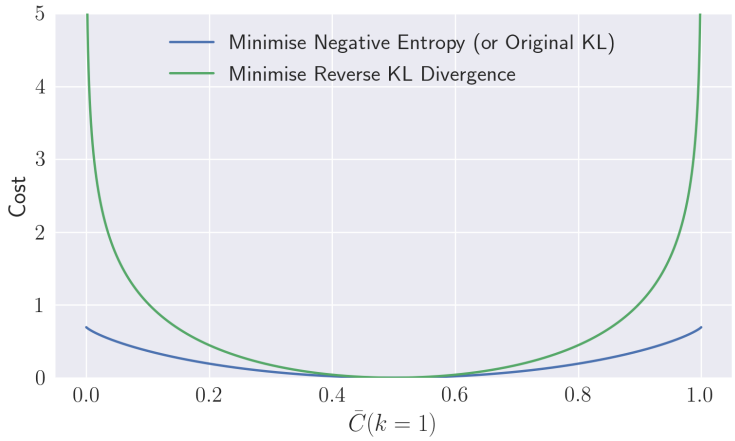


Figure 12: Cost functions implied by minimising the negative entropy of the average distribution, versus minimising its KL-divergence from a flat prior (normalised to minimum cost = 0)

We derived additional results and insight from also viewing an entropy *minimisation* target as a KL divergence minimisation problem. However, as these results are not used in any subsequent analysis or experiments, we defer these derivations to Appendix B.

6.3 Theoretical evaluation of different entropy-related loss functions

The overarching aim of this section is to analyse theoretically the various loss functions we have identified that can be used in a BadGAN-style model. These include the CatGAN-, Improved GAN- and Reverse KL-style loss functions. The primary means for this analysis is investigating the gradients that flow back from the different loss function formulations, and what this means for their learning behaviour. We use this analysis to form hypotheses about which of these options might perform best in the context of a BadGAN-style classification model.

6.3.1 Similarity of Improved GAN and CatGAN approaches

Earlier on in this thesis project, we suspected that the CatGAN and Improved GAN approaches may in fact be numerically equivalent. To address this question, we compare the

CatGAN and Improved GAN approaches. Our approach to SSL introduced in Section 6.1.4 regularised a classifier by minimising entropy on real data points, while maximising entropy on generated data points of low $p(\mathbf{x})$. This approach is equivalent that adopted by CatGAN (except that CatGAN does not *explicitly* try to generate low $p(\mathbf{x})$ points). Improved GAN proceeds in a similar vein, however instead of entropy, it maximises or minimises its $D(\mathbf{x})$ for the real or fake data, respectively.

To analyse the effect of the CatGAN versus Improved GAN losses on training, we derive their gradients with respect to the kth vector of output layer weights $\mathbf{w}_k$. Consider CatGAN's task of maximising entropy on the generated data points. The analogous loss function component for Improved GAN is:

$$\mathcal{L}_{\text{fake}} := -\mathbb{E}_{\mathbf{x}\sim p_G(\mathbf{x})}\left[\log(1 - D(\mathbf{x}))\right] \tag{6.10}$$

$$= -\mathbb{E}_{\mathbf{x}\sim p_G(\mathbf{x})}\left[\log\left(1 - \frac{\sum_{k=1}^{K}\exp(\mathbf{w}_k^T f(\mathbf{x}))}{\sum_{k=1}^{K}\exp(\mathbf{w}_k^T f(\mathbf{x})) + 1}\right)\right] \tag{6.11}$$

$$= \mathbb{E}_{\mathbf{x}\sim p_G(\mathbf{x})}\left[\log\left(\sum_{k=1}^{K}\exp(\mathbf{w}_k^T f(\mathbf{x})) + 1\right)\right] \tag{6.12}$$

Here $f(\mathbf{x})$ gives the feature space, or penultimate layer, representation of $\mathbf{x}$, hereafter abbreviated as $\mathbf{f}$. Where these features are the result of $\mathbf{x}$ sampled from a generator, we shall denote them $\mathbf{f}_G$.

In practice, we approximate this expectation with its value averaged over a minibatch. Its derivative with respect to $\mathbf{w}_k$ for a single generated example is given as:

$$\frac{\partial \mathcal{L}_{\text{fake}}}{\partial \mathbf{w}_k} = \mathbf{f}_G \cdot \frac{\exp(\mathbf{w}_k^T \mathbf{f}_G)}{\sum_{k=1}^{K}\exp(\mathbf{w}_k^T \mathbf{f}_G) + 1} = \mathbf{f}_G \cdot Z C_k \tag{6.13}$$

$$\text{where } Z = \frac{\sum_{k=1}^{K}\exp\left(\mathbf{w}_k^T \mathbf{f}_G\right)}{\sum_{k=1}^{K}\exp\left(\mathbf{w}_k^T \mathbf{f}_G\right) + 1}$$

We can see here Z is a normalisation constant reflecting the size of the first K logits relative to the $K+1$th logit, which is fixed at 1 (as $\mathbf{w}_{K+1} = \mathbf{0}$). Intuitively, Z accounts for how big the error is in calling $\mathbf{f}_G$ a 'real' point, then C_k apportions this error relative to how much each probability contributed to it (note in this section we continue to use C_k to refer to the kth probability according to our classifier, i.e. $C(k|\mathbf{x})$).

Turning now to the CatGAN loss function, which uses entropy, we take the derivative of entropy with respect to $\mathbf{w}_k$.[6] Recall that we showed in the previous section that maximising entropy is equivalent to minimising the KL divergence between the classifier's output probability distribution C and a flat prior q. So equivalently, the derivative of this KL divergence with respect to $\mathbf{w}_k$ is:

$$\frac{\partial D_{KL}\left(C\|q\right)}{\partial \mathbf{w}_k} = \frac{\partial - H(C(\cdot|\mathbf{f}))}{\partial \mathbf{w}_k} = \mathbf{f}_G \cdot \left[(\log C_k + 1)C_k(1 - C_k) - \sum_{i\neq k}^{K}(\log C_i + 1)C_i C_k\right] \tag{6.14}$$

We can see this is quite different to the gradient flowing back from the Improved GAN loss. Thus we can conclude that the CatGAN and Improved GAN approaches do in fact differ numerically, although we argue later in this section that there still are similarities between these approaches.

Since it may be of interest as an additional proxy for entropy, we also take the derivative of the Reverse KL divergence between a flat prior and the estimated probability distribution

[6] Specifically we are looking at the loss function component which aims to maximise entropy on generated data points, but our findings also apply to the entropy minimisation case, which is just the negative of this component.

(Equation 6.9) as:

$$\frac{\partial D_{KL}\left(q \| C\right)}{\partial \mathbf{w}_k} = \mathbf{f}_G \left(C_k - \frac{1}{K} \right) \tag{6.15}$$

To better understand the similarities and differences between these penalty functions, we plot their gradients with respect to the output layer's weights in a very simple setting (see Figure 13 and its caption for a description of this setting).

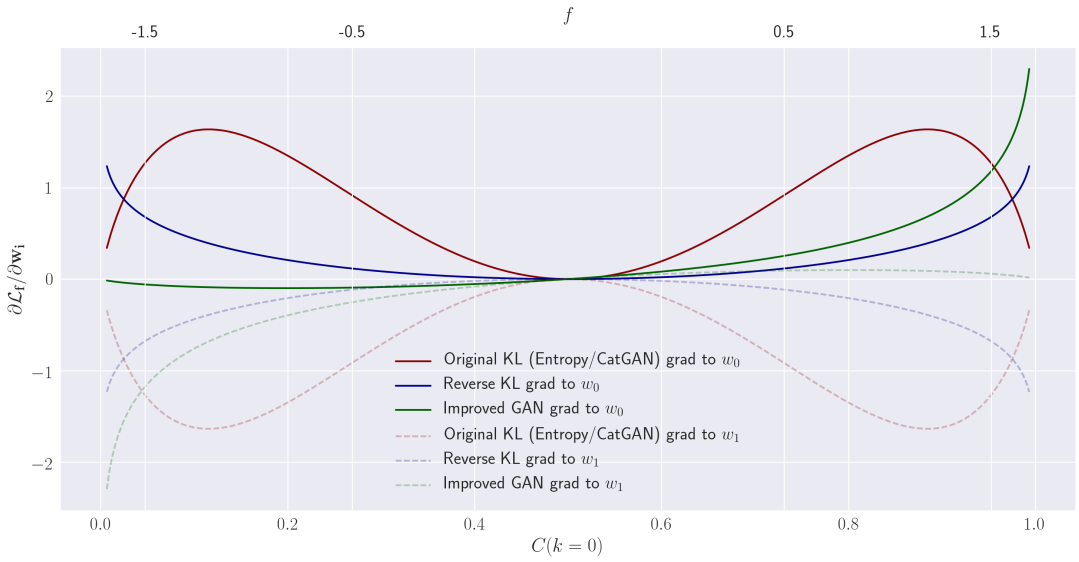


Figure 13: Gradients to $\mathbf{w} := [w_0; w_1]$ implied by the 'Reverse KL,' 'Improved GAN,' and 'Original KL' (i.e., entropy) cost functions as the scalar input f (top axis) is changed. w_0 is fixed at 1.0, w_1 at -1.0, and as such the probability distribution output defined by $softmax(\mathbf{w}f)$ changes as f changes (bottom axis) (note that the Original KL gradient here was multiplied by 30 so that its scale was similar to the gradients of the other two functions).

Immediately obvious is that all three functions' gradients with respect to the weights are zero when entropy is at a maximum. This implies that the end goal of all three functions is maximising entropy (or minimising such, in the case of the opposite loss function component used for real data points). Thus we can conclude that the CatGAN and Improved GAN model's approaches are also similar to some extent, in that both are trying to maximise entropy of the classifier's output in response to generated data points.

The Improved GAN's gradient is unique however, in that it is asymmetric. When f is negative, learning updates with the Improved GAN gradient will increase w_0. In isolation, this would actually reduce entropy, as it would reduce the size of $\exp(w_0 f)$. However, this effect is outweighed by the larger increase in w_1. The overall effect of this is that there is no region where the gradient of the Improved GAN cost function will cause any of the $\exp(w_i f)$ to increase. Despite this, the Improved GAN penalty will still always push the parameters towards entropy maximisation.

The shape of the CatGAN (i.e. entropy) gradient appears to be problematic. We can see that as C_k approaches zero, the entire gradient will approach zero (this is also made clear in Equation 6.14). This is counter-intuitive – we would think that the further away

from our goal of maximum entropy we get, the stronger the gradient pulling us back towards that goal should be. We see this as a critical issue with the CatGAN-style formulation, and expect that as a result its performance may suffer in comparison to the Improved GAN formulation.

Regarding the Reverse KL formulation, we showed in the previous section the steeper slope of its penalty (Figure 12) may be desirable. Its gradients (Figure 13) and their functional form appear to be closer in effect to the Improved GAN formulation than those of CatGAN. Its penalty only depends on the probability C_k however; it is agnostic to the general magnitude of the logits. It is not so clear whether this ambivalence to logit magnitude is desirable. On the one hand, Improved GAN likely constrains the general magnitude of the logits, as these must be compared to the constant logit 0. This may benefit training stability. On the other hand, this constraint might adversely affect modelling capacity by reducing its flexibility. We further discuss this trade-off below.

6.3.2 The CatGAN and Reverse KL approaches may be 'forgetful'

Further potential issues with the CatGAN and Reverse KL loss functions are illustrated by Figure 14. This shows the differences between our three alternative entropy penalties by sketching out the path taken by the weights as the model is updated towards entropy maximisation. We can see that the key difference is that both the KL divergence approaches push both weights towards zero at an even pace, while the Improved GAN approach wants to make both logits as negative as possible, pushing both weights towards the 'correct' value of $-\infty$, but applying a greater force to the 'more incorrect' w_0.

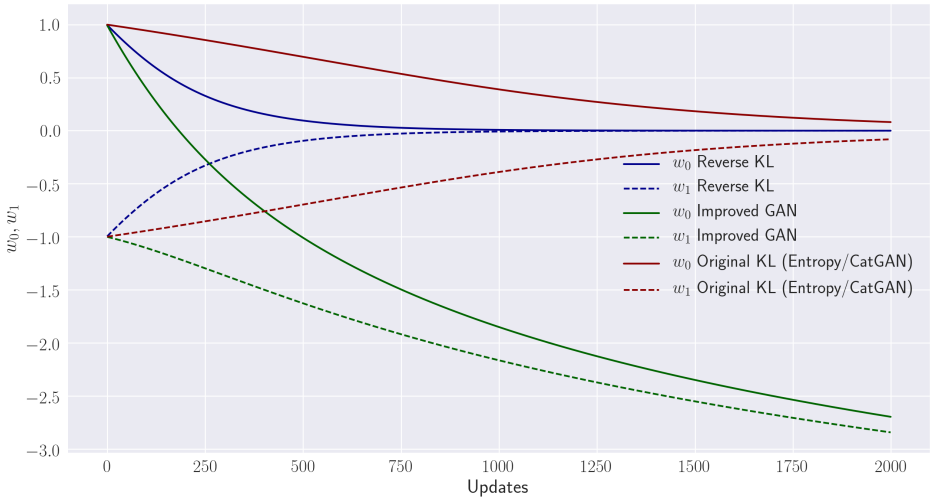


Figure 14: Gradient descent optimisation of $\mathbf{w}$ using the three different loss functions. The gradient of each loss function with respect to $\mathbf{w}$ was calculated with the input f held constant at $f = 1$. The values of the $\mathbf{w}$ vector were updated 2000 times, according to $\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \frac{\partial \mathcal{L}_f}{\partial \mathbf{w}_t}$, where the learning rate α was fixed at 0.01. Through this update mechanism, the weights change such that the 'generated' data point $f = 1$ produces an output probability distribution with increasing entropy. We can see however, that the means by which this goal is achieved, in terms of how the weights are updated, differs substantially between the three loss functions.

Figure 15 highlights the issue with the CatGAN and Reverse KL approaches. This figure shows the effect of each loss function on the output class probabilities as training progresses. The main difference to note is that the Reverse KL and CatGAN formulations converge to $C(k = 0) = 0.5$ much faster than the Improved GAN formulation. We believe this is problematic, as it suggests that the Reverse KL and CatGAN options will cause the model to 'forget' its class preferences more quickly, particularly in regions where there is substantial overlap between the generator and real data distributions. Improved GAN, on the other hand, appears as though it will preserve these class preferences for longer, instead just making the model less confident in these preferences.

We see this potential for 'forgetting' as another drawback for the CatGAN and Reverse KL approaches, as we do not want the model to completely lose its class preferences in areas where there is overlap between the real and generator data distributions. We believe this could lead to higher rates of misclassification in the outer regions of class clusters. In Figure 11 we can already see an effect like this occurring in both of our synthetic experiments (note the high level of uncertainty in the outer parts of clusters, particularly in the Gaussian mixture case). Looking ahead at Figure 16 (bottom-left), we can see that the Improved GAN formulation does not suffer from this issue.

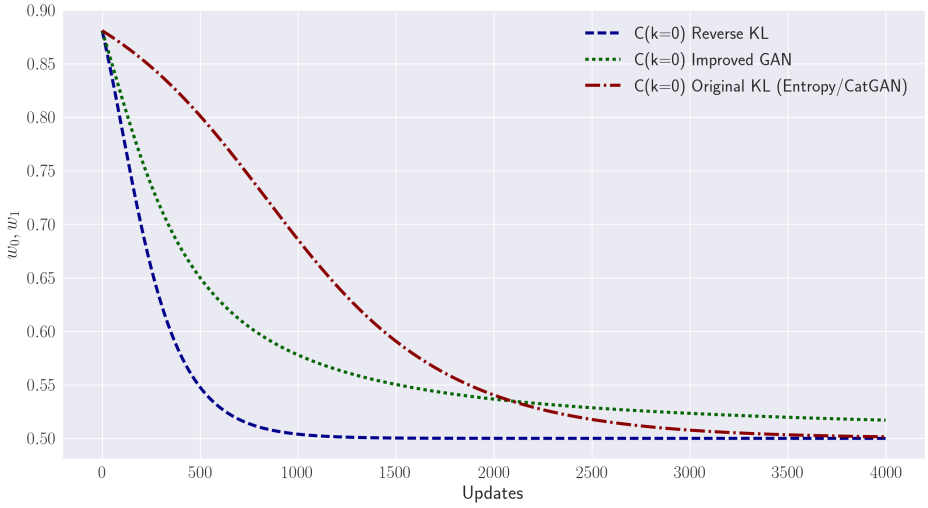


Figure 15: Evolution of the probability of class one as training proceeds as described in Figure 14 (4,000 updates are shown here as opposed to 2,000 in that figure).

6.3.3 Another approach: removing the K+1th class' constraint in the Improved GAN formulation

Zhang and LeCun (2015) have already explored the merits of what we call the Improved GAN and Reverse KL loss function formulations (which they refer to as the 'Background Class' and 'Uniform Prescription' approaches, respectively). Rather than maximising entropy on generated data however, the authors seek to maximise entropy on samples from the '80 Million Tiny Images' dataset that CIFAR-10 is a small subset of [Krizhevsky (2009)], and on the large number of unlabeled images available as part of the STL-10 dataset [Coates et al. (2011)]. In doing this, they assume that the probability of any of these unlabeled images belonging to one of the 10 classes of the labeled portion is practically zero. Discussion over the validity of this assumption aside, the results are still relevant for our purposes.

The authors find that what they refer to as the 'Dustbin' approach achieves the best results across almost all of their tests. This approach is the Improved GAN approach, but without the constraint fixing $\mathbf{w}_{K+1}$ to zero. The Improved GAN approach is also assessed, but is found to perform worse than the Dustbin approach in all but one test. The authors assert that this "is because the extra class is parameterised, which makes it adapt better on the unlabeled samples." The Reverse KL formulation is also found to be inferior to the Dustbin approach; it is less clear though, whether the Reverse KL approach is better or worse than the Improved GAN approach.

6.3.4 Summary

From this theoretical analysis we can draw a number of hypotheses. The first is that the CatGAN (i.e. entropy max/minimisation) approach is likely to not perform as well as other approaches, as the shape of its gradient seems counter-productive in places. Second, it is unclear whether the Improved GAN or Reverse KL formulation will be better, as nothing in our analysis suggests whether constraining the weights per Improved GAN is desirable.

Improved GAN may be preferable however, as it might not 'forget' learned class preferences so quickly, as discussed. Last, the results in Zhang and LeCun (2015) suggest that the Dustbin approach (i.e., Improved GAN without the constraint that $\mathbf{w}_{K+1} = 0$) may be the best overall formulation. We now proceed to evaluate the performance of these four alternatives on our synthetic datasets. Later, in Section 8, we further investigate questions surrounding which of these formulations performs best through larger empirical experiments.

6.4 Experiments with alternative loss functions on synthetic datasets

We have now identified four candidate functionals that could serve as proxies for entropy in our BadGAN-style model. These are the CatGAN approach, the Reverse KL approach, the Improved GAN approach, and the Dustbin approach.

To better understand the differences and potential of these four approaches, we tested each on the synthetic Moons dataset. Specifically, we took our synthetic BadGAN-style approach described in Section 6.1.4, where we minimise entropy on real examples and maximise entropy on generated examples of low $p(\mathbf{x})$, and replaced all entropy terms with that from each of our four candidates. For these experiments, we evenly weighted all of the loss function terms at 1.0 (note that the entropy maximisation penalty term aimed at encouraging the classifier to make even assignment across classes was also included in all experiments). The results (Figure 16) suggest that the CatGAN and Improved GAN approaches work in this setting, while the Reverse KL and Dustbin approaches may be more problematic.

During convergence, the Reverse KL approach gets stuck. We believe this is due to the harshness of its exponentially-shaped penalty. To extend the second class' classification region (the red region) over the misclassified points, it must temporarily incur a greater entropy minimisation penalty on its way. This problem could be alleviated by reducing the weight on the entropy minimisation term to 0.7. Still, in general we find this approach to be quite unstable. Based on this, we expect it may not perform so well on larger datasets.

The Dustbin approach was also problematic. The problem appears to be that without the constraint on the $K + 1$th weight, the resulting model is not as strongly encouraged to be uncertain between the first K classes in the regions of generated data. The result is that there are almost no uncertain regions in the decision space, leading to the misclassification of some of the first class' points. Thus we hypothesise that this approach may also not scale well.

The main conclusion from analysing and performing synthetic experiments on these different loss functions is that there is no one best approach. In fact it appears that the best approach might be to combine the CatGAN and Improved GAN approaches into a single loss function, as in other experimentation we found this approach to be the most robust. In general, based on how training progresses, heuristics can be used to address imbalances and shortcomings in the weightings and formulation of the cost function. The insights gained in this section provide a basis for a number of our research questions, and also give some intuition as to how we might address the problems that may arise during training of a BadGAN-style model.

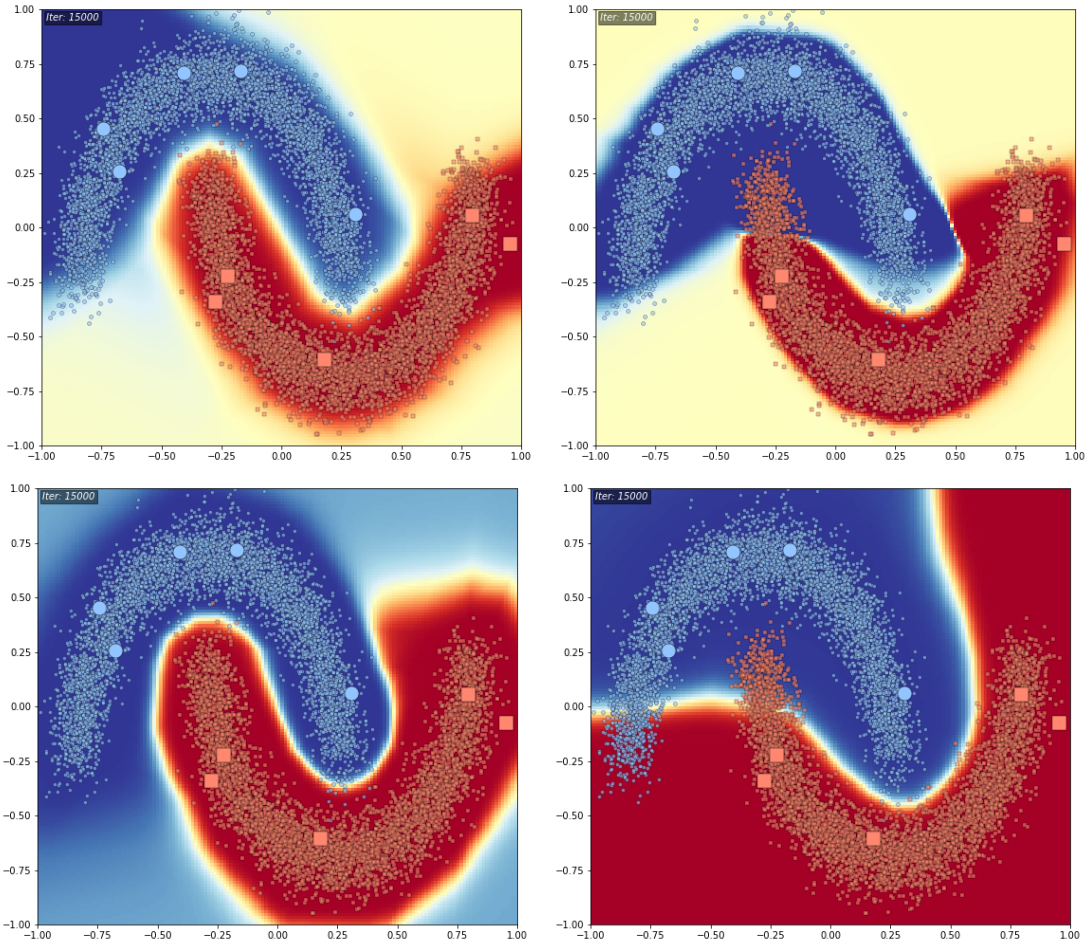


Figure 16: Results from applying the (clockwise from top) CatGAN, Reverse KL, Improved GAN, and Dustbin approaches to the Moons dataset.

For convenience, we repeat the loss functions for these four approaches here:

1a. CatGAN classifier loss $= \mathcal{L}_{\text{xent}} + \mathcal{L}_{\text{ent}_U} - \mathcal{L}_{\text{ent}_G}$
$= \mathcal{L}_{\text{xent}} + \mathbb{E}_{\mathbf{x}\sim p_U}\left[H(C(\cdot|\mathbf{x}))\right] - \mathbb{E}_{\mathbf{x}_G\sim p_G}\left[H(C(\cdot|\mathbf{x}_G))\right]$
1b. CatGAN generator loss $= \mathcal{L}_{\text{ent}_G}$

2a. Reverse KL classifier loss $= \mathcal{L}_{\text{xent}} + \mathbb{E}_{\mathbf{x}\sim p_U}\sum_{k=1}^{K}\log C(k|\mathbf{x}) - \mathbb{E}_{\mathbf{x}_G\sim p_G}\sum_{k=1}^{K}\log C(k|\mathbf{x}_G)$
2b. Reverse KL generator loss $= \mathbb{E}_{\mathbf{x}_G\sim p_G}\sum_{k=1}^{K}\log C(k|\mathbf{x}_G)$

3a. Improved GAN classifier loss $= \mathcal{L}_{\text{xent}} - \mathbb{E}_{\mathbf{x}\sim p_U}\log D(\mathbf{x}) - \mathbb{E}_{\mathbf{x}_G\sim p_G}\log[1 - D(\mathbf{x}_G)]$
3b. Improved GAN generator loss $= -\mathbb{E}_{\mathbf{x}_G\sim p_G}\log[D(\mathbf{x}_G)]$

4a. Dustbin classifier loss: same as Improved GAN, but without the constraint that $\mathbf{w}_{K+1} = 0$
4b. Dustbin generator loss: as above.

7 Research questions and hypotheses

In this section we present a number of research questions that have been motivated by the preceeding analysis presented in this thesis. For each of these, we also posit some hypotheses as to what we believe the answers to these questions are likely to be. In the following section we then describe and perform the experiments used to test these hypotheses, and discuss the results.

RQ7.1 Which loss function formulation is best?

In the previous section, we found that there is no clear best way to achieve low-density separation through minimising entropy on real samples and maximising entropy on generated samples of low $p(\mathbf{x})$. Thus the question remains as to which proxy for entropy is best out of the CatGAN, Reverse KL, Improved GAN and Dustbin approaches. We hypothesised in the previous section that the CatGAN approach in isolation may be deficient due to the shape of the function defining the gradients that are backpropagated from its loss function. Contrary to the findings of Zhang and LeCun (2015), we also found the Dustbin approach may be problematic. Moreover, there was evidence that the Reverse KL formulation may suffer from optimisation difficulties. Thus we hypothesise that in isolation, the Improved GAN formulation will perform best.

RQ7.2 Can the PixelCNN++ model be replaced by some other density estimate?

In Section 5.4 a shortcoming of BadGAN was noted in that it relies on the computationally-expensive PixelCNN++ density estimation model. We wish to investigate how necessary this model is, and in cases where it is necessary, whether it can be replaced by some other approximation of $p(\mathbf{x})$. There are two alternatives to PixelCNN++ that we propose here: the discriminator from a pre-trained GAN and a denoising autoencoder. A motivation for these choices follows.

7.2.1 Discriminator from a pre-trained generative adversarial network

The discriminator from a GAN learns a function that can tell the difference between points from the true data distribution $p(\mathbf{x})$ and points from the generator's distribution. Of course, if the GAN training converges, eventually the discriminator will output $D(\mathbf{x}) = 0.5$ for both real and generated $\mathbf{x}$ as it cannot tell the difference [Goodfellow et al. (2014)]. However we hypothesise that if we stop training before this point is reached (or if the setup lacks the capacity to reach this point), then the discriminator should still output enough information to signal whether a given data point is approximately on the true data manifold or not. Thus we hypothesise that using a discriminator D and its gradient with respect to our BadGAN model's generator parameters might be sufficient to achieve the goal of not producing images above some arbitrary $p(\mathbf{x})$ threshold.

7.2.2 Pre-trained denoising autoencoder

Our other proposed alternative to the PixlCNN++ model is to train a denoising autoencoder (DAE). To train a DAE, we add Gaussian noise to our data points $\mathbf{x}$ and obtain $\tilde{\mathbf{x}} = \mathbf{x} + N(0, \sigma_n^2 \mathbf{I})$. We then train a neural network function $g(\cdot)$ which takes as its argument $\tilde{\mathbf{x}}$ and aims to reconstruct the original vector $\mathbf{x}$ from this (usually progress on this aim is represented by an L2 reconstruction loss). As we generally assume that there is some lower-dimensional manifold on which the data reside, usually this function $g(\cdot)$ contains a bottleneck layer of lower dimension than the input vector. This layer is generally sandwiched between an encoder and decoder network which consist of a number of hidden layers.

Our motivation for the use of a DAE as a proxy for $p(x)$ can be explained both intuitively and theoretically. Intuitively, a DAE should learn a function that can take a noisy image and bring it back to the true data manifold. As such, the reconstruction distance a DAE applies to a sample should in general be proportional to the distance of that sample to the true data manifold. Thus we believe the distance between an input $\mathbf{x}$ and $g(\mathbf{x})$ should be inversely proportional to $p(\mathbf{x})$.

This notion is supported in Alain and Bengio (2014), where it was derived that an optimal denoising function captures the *score* (i.e. the derivative of the log-density with respect to the input) of $\mathbf{x}$:

$$g(\tilde{\mathbf{x}}) = \tilde{\mathbf{x}} + \sigma_n^2 \nabla_{\mathbf{x}} \log p(\mathbf{x}) + \mathcal{O}(\sigma_n^2) \text{ as } \sigma_n \to 0 \tag{7.1}$$

In a blog post on their website [Curious AI Company (2016)], the authors of the Ladder network give a derivation that removes the small noise limit requirement in the above equation by instead expressing the denoiser as a function of the score of the corrupted data distribution $p(\tilde{\mathbf{x}})$. They show that:

$$g(\tilde{\mathbf{x}}) = \tilde{\mathbf{x}} + \sigma_n^2 \nabla_{\tilde{\mathbf{x}}} \log p(\tilde{\mathbf{x}}) \tag{7.2}$$

Since we just added independent Gaussian noise to $p(\mathbf{x})$ to sample $\tilde{\mathbf{x}}$, we know that the modes of $p(\tilde{\mathbf{x}})$ should roughly be the same as $p(\mathbf{x})$ (as the mode of the Gaussian noise distribution occurs at 0, i.e. when we make no change to $p(\mathbf{x})$). Therefore we assume that this second approximation will also be reasonable for estimating the score of $p(\mathbf{x})$. Thus we have that $\nabla_{\mathbf{x}} \log p(\mathbf{x}) \approx (g(\mathbf{x}) - \mathbf{x})/\sigma_n^2$. We can backpropagate this gradient directly to our generator after evaluating it on the points $\mathbf{x}_G$ produced by the generator.

Unfortunately however, this does not provide us with point estimates of the density $p(\mathbf{x})$, which is needed if we want to threshold which generated points to flow this gradient back to, as per the original BadGAN model (Equation 5.8). To get around this, following our intuitive interpretation of a DAE's relation to $p(\mathbf{x})$, we simply use the negative reconstruction distance $-(g(\mathbf{x}) - \mathbf{x}))$ to approximate $p(\mathbf{x})$. In practice, we use the 10th percentile of this negative reconstruction distance on the training data to obtain our threshold.

RQ7.3 Do the generated examples actually contribute to feature learning?

In the BadGAN model, we look to generate data points $\mathbf{x}$ that are of low $p(\mathbf{x})$ but that appear in our model's feature space. Ostensibly, the aim of this is to encourage low-density separation in the feature space. However, in general this feature space exists at or towards the end of a feedforward neural network. Given the good performance of this model and its close relatives in SSL tasks, it would appear that the generated images are doing something more than just helping the final layer to learn correct decision boundaries in the feature space; rather, it seems that these images contribute in some way to the learning of more useful features in earlier layers.

Our hypothesis is that the generated images do indeed also assist earlier layers in the classifier network to learn more useful features. In the initial stages of training, when the generated images are very noisy, we expect the effect on the parameters to be similar to L2 regularisation, per Bishop (1995). Later, as the generated images become more realistic, we expect that they will assist through a 'sharpening' of the learned features, as the model must learn to distinguish between the blurry images often produced by a generator and the sharper images of the true dataset.

RQ7.4 Is VAT or InfoReg really complementary to BadGAN?

In Dai et al. (2017) and Miyato et al. (2017), the authors suggest that their approaches – BadGAN and VAT respectively – are orthogonal to one another, and thus could be combined

to provide performance improvements. Our hypothesis here is that there may be some synergies, but as these methods are both based on enforcing low-density separation, their benefits may in fact overlap to a large extent and we may see little improvement through their combination. Given its similarity to VAT, we wish to also assess whether InfoReg could play a similar role to VAT here. Also more generally we are interested to see how well InfoReg competes with VAT, but as mentioned this analysis is deferred to Appendix A.

8 Experiments

To investigate our research questions, we conducted experiments on the PI-MNIST-100 and SVHN-1k datasets (refer to Box 1 for a description of these datasets). We divide this section by dataset. For each dataset we first give an overview of our general experimental setup, describing the type of model that was used and other aspects of the training procedure. We then proceed to present and discuss results from the experiments, in response to each our research questions.

8.1 PI-MNIST-100

8.1.1 Experimental setup

We now describe the model setup for PI-MNIST-100 in detail. The basic BadGAN setup is as follows. We have a classifier C that is a multilayer perceptron (MLP) network which takes an input $\mathbf{x}$ and outputs a probability distribution over the K classes $C(\cdot|\mathbf{x})$. In line with Rasmus et al. (2015) we adopt hidden layer sizes of [1000, 500, 250, 250, 250]. The input to this MLP is a flattened MNIST digit of shape $28 * 28 = 784$, and its output is $K = 10$ units. Leaky ReLU [Maas et al. (2013)] activations are used at every layer. Input noise is added to the image, as well as to the hidden layer activations. Batch normalisation is not used. The weights for each layer use the initialisation for ReLU layers prescribed by He et al. (2015). Dropout with a drop probability of 0.3 is added after the hidden layer activations of the first four layers. No batch normalisation or L2 regularisation is added to the parameters of either the generator or discriminator networks.

Our generator is also a neural network. This takes a 128-dimensional vector $\mathbf{z}$ as input. A single $\mathbf{z}$ is produced by sampling from a multivariate Gaussian distribution with 128 dimensions with zero mean and unit diagonal covariance. The generator network has three hidden layers of size [500, 500, 1000]. The output of the generator network is then passed through a sigmoid non-linearity to give $\mathbf{x}_G$ (as the input MNIST digits are rescaled to be between 0 and 1).

In terms of loss function, the starting point for our model is equivalent to CatGAN. For convenience we restate that loss function here:

$$\max_G \min_D \mathcal{L}_{\text{xent}} + \mathcal{L}_{\text{ent}_U} - \mathcal{L}_{\text{ent}_G} \tag{8.1}$$

where:

$$\mathcal{L}_{\text{xent}} = \mathbb{E}_{\mathbf{x},y\sim p_L(\mathbf{x},y)} \left[\sum_{k=1}^{K} \delta(y=k) \log C(k|\mathbf{x}) \right] \tag{8.2}$$

$$\mathcal{L}_{\text{ent}_U} = \mathbb{E}_{\mathbf{x}\sim p_U} \left[H(C(\cdot|\mathbf{x})) \right] \tag{8.3}$$

$$\mathcal{L}_{\text{ent}_G} = \mathbb{E}_{\mathbf{x}_G\sim p_G} \left[H(C(\cdot|\mathbf{x}_G)) \right] \tag{8.4}$$

That is, the discriminator aims to maximise entropy on generated samples, minimise entropy on unlabeled samples, and minimise categorical cross-entropy of its predictions and the labeled samples. To encourage an even class assignment, we also add the $H\left(\bar{C}\right)$ term from Equation 6.3 to maximise entropy on the average probability distribution of unlabeled samples. The generator's loss then encourages it to minimise entropy on its generated samples, according to the discriminator's output.

A batch size of 64 labeled examples, 256 unlabeled examples, and 256 generated examples is used in each training iteration. In each iteration, the generator is trained for one step, and then the discriminator is trained for up to five steps. The Adam [Kingma and Ba (2014)] optimiser is used, with a learning rate of 0.0001, β_1 of 0.5 and β_2 of 0.9. We found lowering the learning rate and momentum from Adam's stronger defaults in this manner to be crucial for MNIST performance. Models are trained for 150 epochs, with the learning rate linearly

decayed to zero starting after the 130^{th} epoch (an epoch here is defined as N/B, where N is the number of unlabeled input images in the training set and B is the number of unlabeled data points in each batch).

For all experiments, we focused on the setting where all but 100 labels (10 labels per class) from the training set are made available. The last 10,000 examples are held out from the training set as a validation set, which is used for hyperparameter tuning. Final model performance is then evaluated on the standard MNIST test set, which also consists of 10,000 labels. All results presented in this section are averages over six different random seeds.

8.1.2 Effectiveness of different proxies for entropy

Comparison of CatGAN, Reverse KL, Improved GAN and Dustbin approaches. After some parameter tuning, our initial model setup – equivalent to CatGAN – achieved a 3.56% error on the test set (Figure 17). This is somewhat higher than the 1.91% error reported in the CatGAN paper [Springenberg (2015)]. One crucial aspect of getting to this level of performance included having the discriminator train five times per iteration. Presumably this was due to the quality of generated images becoming too realistic before the discriminator could be sufficiently regularised by the poorer quality images. Another key aspect was using a standard deviation of 0.3 for the noise added to both the input image and at all hidden layers.

We then turned to our hypothesis that CatGAN's entropy loss' functional form is deficient. We replaced all entropy minimisation/maximisation targets with the 'Reverse KL' form of this function. That is, any reference to entropy $H(p) = \sum_{k=1}^{K} -p_k \log p_k$ in the model was replaced by $H'(p) = \sum_{k=1}^{K} \log p_k$. In line with our expectations, we find this functional form performed slightly better; it achieved an average error on the test set of 2.48% (Figure 17). Some tuning of the weight in front of H' was required to get this result; we find that pre-multiplying any instances of this cost by 0.1 to produce the best results on the validation set.[7] We believe the improvement is due to the more rational shape of the gradient of the Reverse KL loss function (recall Figure 13). We also believe the cost weight tuning is necessary here as the cost from the Reverse KL form tends to be much larger due to the steeper slope of this function.

We then replaced the CatGAN entropy minimisation/maximisation terms with the corresponding terms used in Improved GAN. We found that this immediately improved performance, dropping the error rate to 1.16% (Figure 17).

Regarding the Dustbin approach, our expectation was that its increased parameter flexibility would further improve performance. Mainly however, we find this approach results in training stability issues. Inspection of our model revealed that this instability was caused by very large logits leading to overflows. This problem could be mitigated with L2 regularisation, but even with this at a seemingly-optimal setting, training still collapses around 50% of the time (Figure 17). In instances where training does not collapse, accuracy is no better than the Improved GAN approach. Given these results, we concluded that the Dustbin approach is too unstable in the context of BadGAN-type models, and discontinued its use.

As training progresses, the cross-entropy cost on the labeled dataset (Figure 18) provides a signal as to why the Improved GAN performs better in this instance. The cross-entropy cost is much lower under the Improved GAN penalty. The translation of this into stronger performance on the test set highlights the importance of correctly classifying the limited labeled samples in training this SSL model.

Figure 18 also supports our earlier hypothesis, stated in Section 6.3. This hypothesis suggested that the way in which the CatGAN-style approaches cause the weights to be updated could cause the model to more quickly 'forget' its class preferences, particularly

[7] To be fair, we also tuned this weight hyperparameter for the original CatGAN model, but found that a weight of 1.0 was best anyway.

Figure 17: Accuracy on the test set over time when training a CatGAN-style model on the PI-MNIST-100 dataset, using the CatGAN, Reverse KL, Improved GAN and Dustbin formulations of the entropy losses.

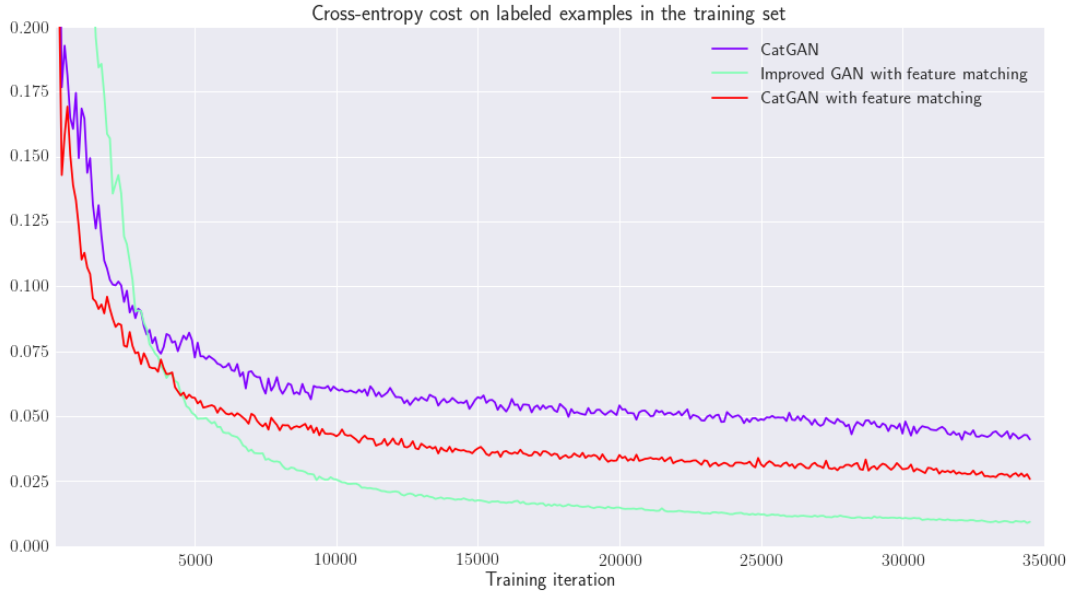


Figure 18: Progress on cross-entropy loss on 100 labeled data points of PI-MNIST-100 whilst training with the CatGAN, Reverse KL, CatGAN with Feature Matching, and Improved GAN (with feature matching) loss formulations.

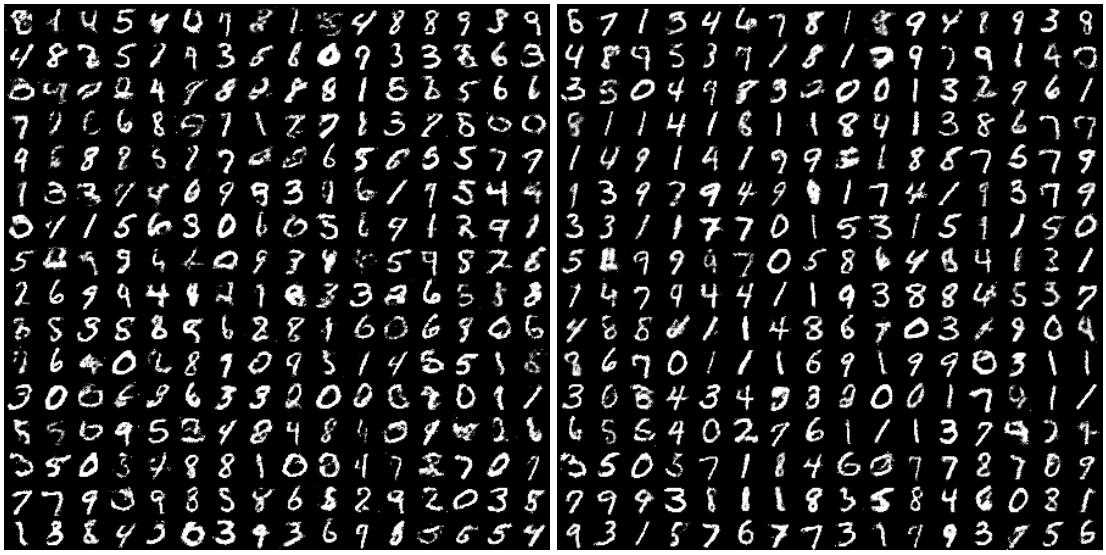


Figure 19: Sample of images produced by Improved GAN generator with feature matching (left) versus without feature matching (right), i.e. with $D(\mathbf{x})$ in the generator's cost function instead. Note that those produced without feature matching appear sharper.

in regions where the true and generative data distributions overlap. If the model is indeed 'forgetting,' then the probabilities it assigns to the correct classes will be lower, and this will increase categorical cross-entropy loss. It also appears adding feature matching to the CatGAN reduces this effect. This also conforms to our hypothesis, if feature matching causes the generative distribution to have less overlap with the true data distribution. We argue in the next paragraph that this is likely the case.

Feature matching versus entropy minimisation as the generator loss
We also find changing the CatGAN generator's loss function to use feature matching (instead of trying to minimise the entropy of its generated images) leads to substantial performance gains, dropping the test set error rate to 2.0% (Figure 20). Inspection of the generated images reveals that feature matching led to slightly worse-looking samples (Figure 19). Given this, we believe the performance improvement here is due to the resulting generator being closer to the desired 'complement generator' that produces low probability images that still lie in the feature space of the discriminator.

To see whether these benefits of feature matching carry over to the Improved GAN setting, we took this CatGAN model with feature matching and changed its discriminator's entropy losses to the equivalent components of the Improved GAN loss. This resulted in a further drop in the test set error rate to 1.2%. Surprisingly this is still slightly higher than the 1.16% error we attained when using the Improved GAN loss without feature matching. Inspection of the images produced by this Improved GAN run still suggested that those with feature matching were subjectively worse. This runs counter to our previous argument that grainier images lead to better performance. Looking at the progression of test set accuracy over time (Figure 20) provides a potential explanation. The model without feature matching learns much faster, so presumably the higher-quality generated images that are produced earlier on in training are more useful in this instance, and the benefits from this remain in the model in the long run.

Summary

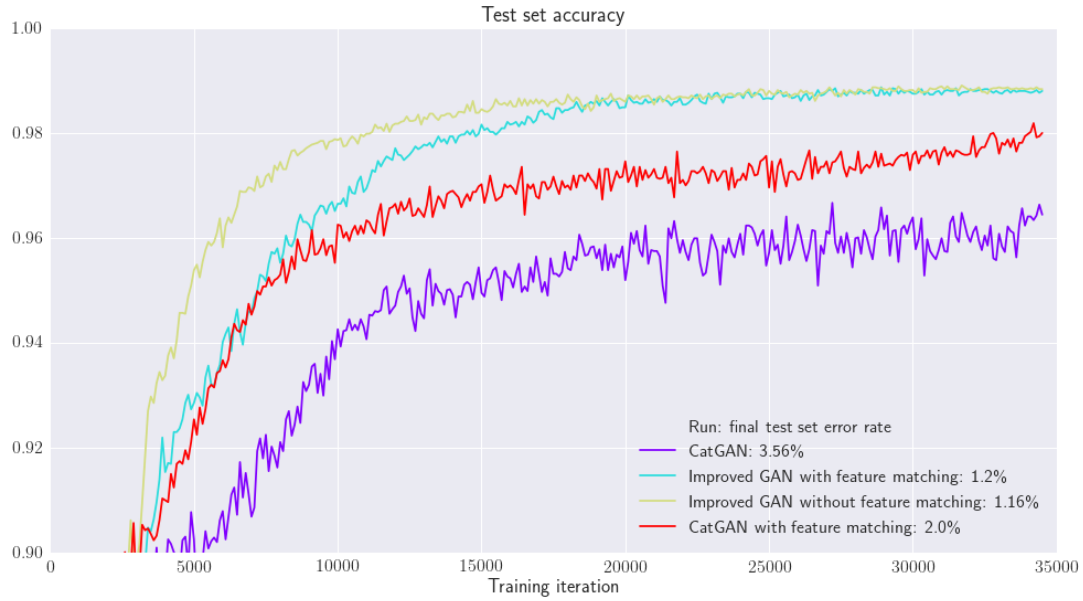


Figure 20: Accuracy on the test set over time when training a CatGAN-style model (i.e., where the generator aims to minimise the entropy of its fake images, and the classifier aims to maximise the entropy of these, and minimise the entropy of real images) on the PI-MNIST-100 dataset, using the CatGAN (or original KL), Reverse KL, Improved GAN and Dustbin formulations of the entropy losses.

In response to RQ7.1, our findings on the Improved GAN versus CatGAN loss functions support our hypothesis that the Improved GAN loss function is better-formulated for BadGAN-style learning. It appears that this owes at least in part to the phenomenon of the CatGAN-style loss causing the model to more quickly 'forget' its class preferences in regions of uncertainty, as we discussed in Section 6.3. This is evidenced by the Improved GAN's better performance on the cross-entropy loss of the labeled data (along with evidence of this phenomenon in our synthetic experiments). Our experiments are less conclusive with regards to whether feature matching or entropy minimisation is a better loss function for the generator to try and minimise. While adding feature matching to CatGAN substantially improved its performance, turning it off in the Improved GAN actually led to slightly better results.

8.1.3 Potential for replacing PixelCNN++ model with a DAE or discriminator from a GAN

To address RQ7.2, we take our Improved GAN with feature matching model and replace the generator loss function with that prescribed by the BadGAN model. For convenience we repeat that equation here:

$$\mathcal{L}_{\text{gen}} - H(p_G) + \mathbb{E}_{\mathbf{x}\sim p_G} \log p^*(\mathbf{x}) \delta \left[p(\mathbf{x}) > \epsilon \right] + \mathcal{L}_{\text{FM}} \tag{8.5}$$

As described in Section 5.4, to descend the gradient of this equation, we need a proxy for $H(p_G)$ and $p(\mathbf{x})$ (as we will use the same function for $p(\mathbf{x})$ and $p^*(\mathbf{x})$). For $H(p_G)$, we used the pull-away term, described in Section 5.4, with a weight of 1.0. For a model providing point estimates and gradients of $p(\mathbf{x})$, we have three approaches: a PixelCNN[8], the discriminator from a GAN, and a DAE.

We pre-trained each of our three $p(\mathbf{x})$ model choices on the training dataset.[9] We then loaded these pre-trained models during training time and fixed their parameters. We are then able to query these for point estimates, and gradients with respect to our generator's parameters, of $p(\mathbf{x})$. Like in the original BadGAN paper [Dai et al. (2017)], we used the 10th percentile value of $p(\mathbf{x})$ according to the particular model being used to give the threshold $p(\mathbf{x})$ value ϵ. The weight on the 'generated data should be low probability' loss term ($\mathbb{E}_{\mathbf{x}\sim p_G} \log p^*(\mathbf{x}) \delta \left[p(\mathbf{x}) > \epsilon \right]$) was tuned between 1 and 10000, in multiples of 10, to find the optimal weight.

The results (Figure 21) suggest that one of our proposed approaches - that based on a DAE - may work just as well as a PixelCNN model. That said, the improvement provided by either model is not particularly stark. Adding the PixelCNN model (and pull-away term) to the Improved GAN reduced its average error by just 0.03% – from 1.20% to 1.17%. The improvement due to the DAE approach was more significant, reducing the error to 1.06%. Still, both these results trail the PI-MNIST-100 scores reported in the BadGAN paper of 0.8%.

As also shown in Figure 21, the approach using the discriminator from a pre-trained GAN as a $p(\mathbf{x})$ proxy was unstable. We are not entirely sure what the exact cause of this is, though we suspect the similarity of the training signal from the pre-trained discriminator, and that from the classifier (due to the fact that both use GAN training processes) may be interfering with the dynamics of the already-volatile GAN training procedure. Given this instability, we conclude that this approach is not a good candidate as a $p(\mathbf{x})$ model.

[8] We decided to use PixelCNN [Oord et al. (2016)] instead of PixelCNN++ in our experiments, as it is more easily implemented, and should have the same optimum, as it is essentially a less-efficient, more precise form of PixelCNN++.

[9] $p(\mathbf{x})$ model architecture details were as follows. Our DAE and GAN models used the same encoder/decoder or discriminator/generator architecture as the classifier/generator from the BadGAN model being fit. The Improved WGAN Gulrajani et al. (2017) method was used for training GANs. Models were fit until realistic-looking images were being generated (in the GAN case), or until the reconstructions looked quite reasonable (DAE) case. Our PixelCNN model consisted of nine convolutional layers with ReLU activations, kernel sizes of 5 and strides of 1.

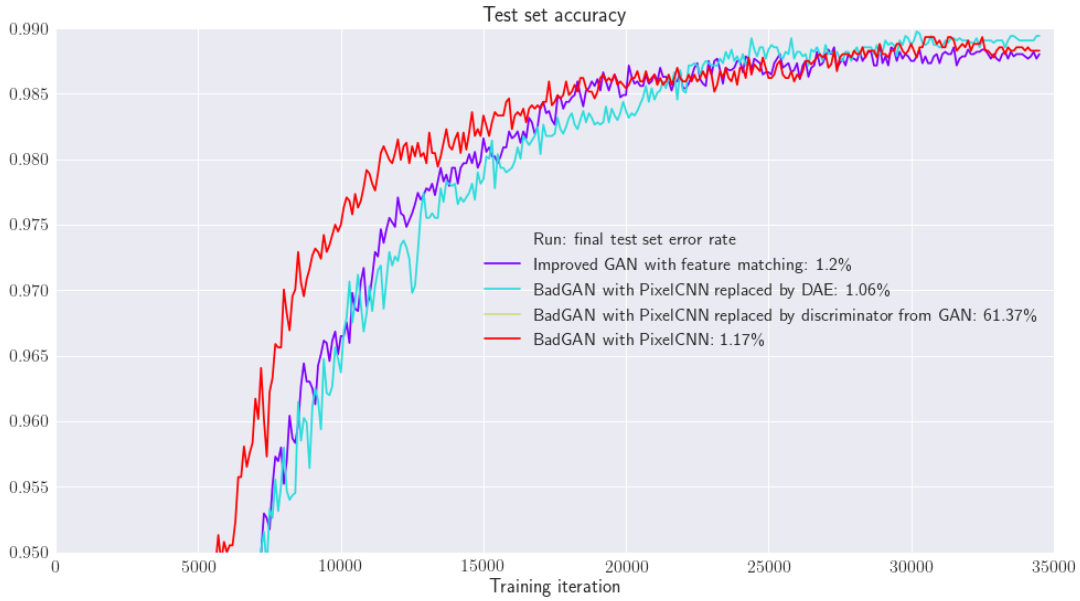


Figure 21: Performance of different proxies for $p(\mathbf{x})$ in the BadGAN model, including a PixelCNN model, a DAE, the discriminator from a GAN (whose performance was too low to be shown at this scale), plotted against the Improved GAN model's performance.

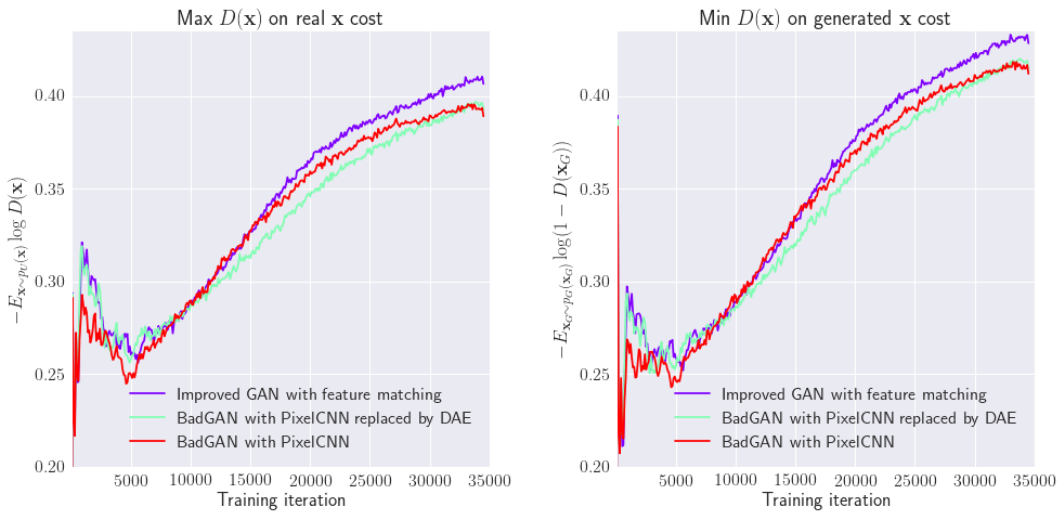


Figure 22: Improved GAN and BadGAN model variations' $D(\mathbf{x})$ costs on the real and generated data. A higher cost indicates that the model is making more errors in distinguishing between the real and fake images.

In terms of speed, our DAE-based approach has a clear advantage over a PixelCNN-based approach. Training with the DAE $p(\mathbf{x})$ model took around one hour and ten minutes[10] – only ten minutes longer than the original Improved GAN model. Training with the PixelCNN took almost five times longer, at five hours and 13 minutes. Moreover, pre-training the PixelCNN model to an appropriate level for MNIST takes around 8 hours, while pre-training the DAE takes less than one hour.

We are able to further explore the efficacy of the DAE versus PixelCNN approaches to modelling $p(\mathbf{x})$ by looking at their impact on the model's ability to distinguish real and fake examples. We would expect that if the additional cost term in the BadGAN model indeed ensures the model produces images of lower $p(\mathbf{x})$, then the discriminator should have an easier time distinguishing between real and generated images.

Looking at the $D(\mathbf{x})$ cost the model scores on real versus fake images gives an indication of how 'easy' it is for the model to separate real from fake. We can see (Figure 22) that over the first 5000 training iterations, the model improves its real/fake cost on both the real and fake data; the $D(\mathbf{x})$ cost reduces for both real and fake $\mathbf{x}$ over this period. This indicates that from its initialised state, the model first gets better at discriminating real from fake. After around 5000 iterations however, the generated images start to become more realistic, to the point that the model's ability to distinguish them from real images becomes constantly worse. This is indicated by the fact that the $D(\mathbf{x})$ costs start going up for both real and generated images.

However, we can also see (Figure 22) that when we add the PixelCNN or DAE model to prevent the $p(\mathbf{x})$ of generated images from becoming too high, then the rate of increase in the $D(\mathbf{x})$ costs is reduced. This indicates that both of these approaches are achieving their desired effect of making the generated images less realistic. We can also see that this effect is stronger in the case of the DAE, which likely explains that approach's better performance.

At this point, a reader might suggest that we should stop training of the generator after this turning point in the $D(\mathbf{x})$ cost, or at least slow it down substantially, and use this as

[10] Training speeds are based on runs on a single NVIDIA Geforce TITAN X GPU.

an easier method of ensuring the generated images do not become too realistic. We did try such an approach, however we found that this substantially decreases performance.

We find that the classifier still learns a lot from generated images that look quite realistic. We posit that there are two reasons for this. The first is simply that encouraging the model to maximise entropy (or similarly, to minimise $D(\mathbf{x})$) on fake images that are so realistic that they often lie on the true data manifold, regularises the model by telling it to not be 'too confident.' Such a technique has been shown in Pereyra et al. (2017) to in and of itself be capable of improving model performance.

The second reason we believe that high quality generated images are still important for regularising our model can be explained as follows. Our discriminator cost component for distinguishing true/fake data is:

$$\mathcal{L}_{\text{true/fake}} = -\mathbb{E}_{\mathbf{x}\sim p_U(\mathbf{x})}\left[\log D(\mathbf{x})\right] - \mathbb{E}_{\mathbf{z}\sim p_z(\mathbf{z})}\left[\log(1 - D(G(\mathbf{z})))\right] \tag{8.6}$$

In practice, we modify this to the following, as it produces stronger gradients earlier in training and has the same fixed point in terms of the GAN training dynamics [Goodfellow et al. (2014)]:

$$\mathcal{L}_{\text{true/fake}} = -\mathbb{E}_{\mathbf{x}\sim p_U(\mathbf{x})}\left[\log D(\mathbf{x})\right] + \mathbb{E}_{\mathbf{x}_G\sim p_G(\mathbf{x}_G)}\left[\log D(\mathbf{x}_G)\right] \tag{8.7}$$

Note we have also switched our notation here to use p_G – the probability distribution defined by sampling $\mathbf{z} \sim p_{\mathbf{z}}(\mathbf{z})$ and placing it through the generator $G(\mathbf{z})$.

Now we imagine a situation where our generator distribution p_G is such that with probability q it samples from an ideal complement generator distribution p_C, and with probability $(1 - q)$ its images are too realistic, and thus are drawn randomly from the true (equal to unlabeled) data distribution p_U. This 'incorrect' discriminator cost becomes:

$$\mathcal{L}_{\text{incorrect}} = -\mathbb{E}_{\mathbf{x}\sim p_U(\mathbf{x})}\left[\log D(\mathbf{x})\right] + q\,\mathbb{E}_{\mathbf{x}_G\sim p_C(\mathbf{x}_G)}\left[\log D(\mathbf{x}_G)\right] + (1-q)\,\mathbb{E}_{\mathbf{x}\sim p_U(\mathbf{x})}\left[\log D(\mathbf{x})\right] \tag{8.8}$$

$$= q * \mathcal{L}_{\text{true/fake}} \tag{8.9}$$

Hence, we can see that if our generator becomes 'too good' in the sense that sometimes it randomly produces images from the true data distribution, then in expectation we can just divide the true/fake loss component by q to correct for this.
In practice, there are two problems with this notion:

1. We do not know q;
2. If q becomes very small, the training signal from the very few complement generator samples will be too weak to learn from in practice.

We can gain some insights from this exercise, however. First is that, as our generator gets to a point where it is producing quite realistic images, it may be wise to start increasing the weight on the true/fake cost. We experimented with this, and found that it can provide some benefit, though we leave a more rigorous evaluation of this to future work.

Second, this exercise gives some explanation as to why the model still learns at a point when most of the generated images look quite realistic. What we have just shown is that, even if only some small proportion of the generated images are off the manifold, the model can still learn from these images.

We believe that a lot of the model regularisation afforded by the BadGAN-style approach stems from generated images that are very close to the true data manifold, as this contributes to the classifier learing 'sharper' features, as described in RQ7.3. As such, it is important to keep training the generator until a lot of its images do look realistic, as the generated images that remain close to, but are still off the true data manifold are informative to our classifier.

8.1.4 Extent to which generated images contribute to feature learning

RQ7.3 questions whether the generated images contribute to feature learning in earlier layers of the classifier. This question is important, as if the generated images' only purpose is to regularise the linear classification function learned by the weights connecting the feature space to the output layer, then theoretically we should be able to achieve the same results as the BadGAN model by directly generating points in the feature space. This presumably would be a much easier task.

To test the hypothesis that the generated images do in fact meaningfully contribute to higher-level feature learning, we stopped the gradients from flowing back from the feature space activations to earlier layers in the classifier. The results clearly confirmed our hypothesis; without these gradients, model accuracy on the validation set was at first poor, and then became null due to numerical stability issues. Analysis of the model revealed that, without the classifier being able to adjust its features in response to generated images, it started producing very large logits for both real and generated images, which eventually led to the observed numerical stability issues.

It is possible that stopping the gradient in this fashion led to stability issues due to it causing problems with the GAN training dynamics, rather than this being a problem specifically due to the classifier not being able to learn higher-level features. We could further investigate this by having the generator instead try to directly produce points in the feature space of the classifier. However we believe what we have already seen is strong evidence that generating images, rather than features, is crucial. Thus we leave deeper investigation of this research question to future work, but conclude that it is likely that the generated images play an important role in learning higher-order classifier features.

8.1.5 Complementarity of VAT and BadGAN

To assess RQ7.4, we disable the generator and all cost components associated with it, and run our best-tuned setup with the virtual adversarial training (VAT) cost added to the discriminator loss. With this VAT-only setup, after tuning the VAT-specific parameters we achieve an average test set error of 1.08% (Figure 23); somewhat better than that reported in the original VAT paper.

To investigate the supposed orthogonality of VAT and Improved GAN, we take our Improved GAN model and add the VAT cost with the best-performing parameters from the runs described above. VAT provides some additional benefit to the Improved GAN's final test set accuracy, which now reaches an average of 1.02% (Figure 23). In our view however, this improvement is not substantial enough to suggest that these two approaches are clearly orthogonal, given how well each performs in isolation. VAT also provides a small boost to the performance of our BadGAN model (using a DAE instead of a PixelCNN model for $p(\mathbf{x})$), and replacing VAT with InfoReg slightly furthers this improvement.

The authors of BadGAN assert that "VAT aims to enforce a smooth function while we aim to leverage a generator to better detect the low-density boundaries" [Dai et al. (2017)]. Our belief is that while VAT does indeed encourage a smooth function, this aspect is not the dominant means by which it achieves SSL. Other approaches, like that in Sajjadi et al. (2016) leverage the importance of a smooth function much more broadly, by enforcing model invariance to diverse input and model perturbations. VAT on the other hand, only promotes invariance in the specific regions covered between the unlabeled data and its virtual adversarial examples. We believe VAT's primary means of achieving SSL is actually through low-density separation. As described in Section 4.2.2, similar to InfoReg VAT achieves such by pushing the decision boundary away from regions of high density.

Given BadGAN also has the goal of encouraging low-density separation, we actually see it as quite similar to VAT. What differs is the means by which each of these method achieve such separation. VAT (and equivalently InfoReg) approaches low-density separation by propagating the decision boundary out from regions of high density. BadGAN instead

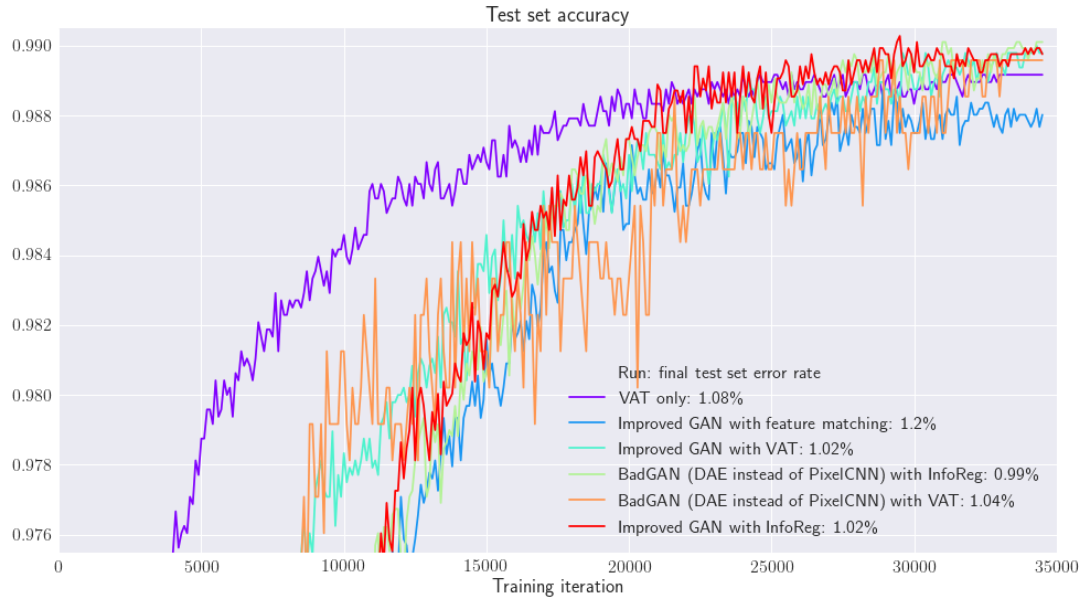


Figure 23: Test set accuracy during training of a VAT-only model and an Improved GAN- or BadGAN-with-VAT model.

produces points in low-density regions, and then encourages the model to place its decision boundary at these points.

Given learning to classify the MNIST dataset is a relatively easy task, we believe that these methods' goal of low-density separation is more or less optimised through either method. Thus, adding VAT to a BadGAN model (or vice versa) already achieving low-density separation will add very little to performance. When scaling up to harder datasets however, we believe there may still be benefits from combining VAT and BadGAN, as neither may be capable in isolation of achieving an optimal low-density separation on more difficult learning tasks. By combining the two paths to the goal of low-density separation, the overall effect on more complex datasets might be to get the overall model closer to that goal than either method in isolation.

To summarise, our response to RQ7.4 is that VAT and BadGAN are not orthogonal. Rather, as both encourage low-density separation they are likely quite similar. From our experiments and discussion, we conclude that benefits of combining these two methods would likely be more limited than their authors suggest. In light of this, we in fact view VAT and BadGAN more as substitutes than complements.

8.2 SVHN-1k

8.2.1 Experimental setup

The modelling setup used for SVHN-1k is largely similar to that used for PI-MNIST-100. The classifier architecture was replaced by a CNN, which is identical to the "Conv Small" model used in the VAT paper [Miyato et al. (2017)]. This model is originally from the Improved GAN paper [Salimans et al. (2016)], and was re-used in BadGAN [Dai et al. (2017)]. The main difference is that Improved GAN and BadGAN both use weight normalisation

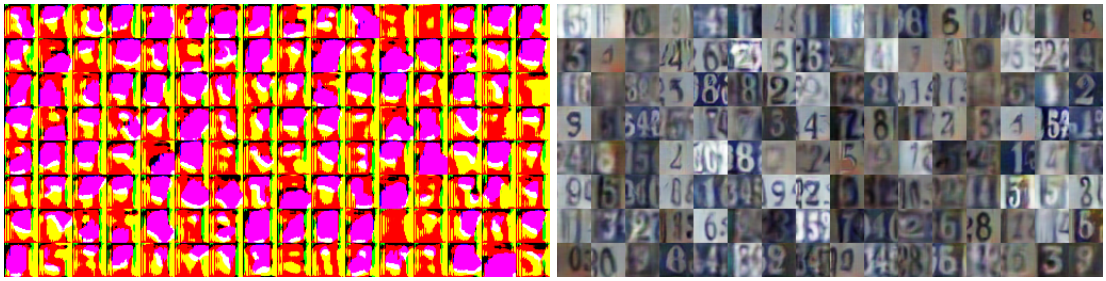


Figure 24: Images generated by our Improved GAN model on SVHN using the method described in the Improved GAN paper (without WeightNorm) (left), versus those generated using feature matching at multiple layers (right).

(WeightNorm) [Salimans and Kingma (2016)], whereas our approach and that in the VAT paper uses batch normalisation [Ioffe and Szegedy (2015)]. Moreover, in the BadGAN paper spatial dropout [Tompson et al. (2015)] is used, whereas we use ordinary dropout.

Specific architecture details of our model can be found in the VAT paper. Briefly though, it consists of 9 convolutional layers, in groups of 3 layers. At the end of each group, a stride of 2 is used to halve the width and height of the feature maps, and then dropout is applied. The [kernel sizes, stride, number of output channels] of each convolutional layer are as follows: →[3,1,64]→[3,2,64]→[3,1,128]→[3,1,128]→[3,2,128]→[3,1,128]→[1,1,128]→[1,1,128]. The final convolutional layer is followed by global average pooling, giving us a vector of length 128. This is then connected to the output layer of 10 units (classes) by a single dense layer, which is followed by a softmax to obtain class probabilities. Leaky ReLU activations with a leakage parameter of 0.1 are again used.

Our generator architecture is similar to that used in the BadGAN paper [Dai et al. (2017)]. It consists of a large dense layer from a Gaussian-distributed $\mathbf{z}$ (of size 128) to the first set of feature maps (of size $4 * 4 * 512$), followed by 3 convolutional layers with [kernel sizes, stride, number of output channels] of [5,1,256]→[5,1,128]→[5,2,3]. A resize operation is used before each convolutional layer, such that the width and height of the feature maps go from 8 to 16 to 32 (following advice from Odena et al. (2016) we used nearest-neighbour interpolation for this operation). A tanh nonlinearity at the output layer takes us back to the space of the input images.

Other differences from the PI-MNIST-100 model are that no noise is added to the input or hidden layer activations. The Tensorflow-default Adam learning rate parameters are also kept. Last, batches of size 32/128/128 (labeled/unlabeled/generated) are passed through the model in each of the 400 training steps which comprise each of up to 200 epochs (with the learning rate linearly decayed to zero after 150 epochs).

8.2.2 Effectiveness of different proxies for entropy

In the case of SVHN-1k, it is difficult for us to respond to RQ7.1 (which regards finding an optimal loss function formulation), as we have not been able to find a clear, significant benefit from using generated images to regularise our classifier under any loss function (Figure 25). We find that mimicking the setup described in the Improved GAN or BadGAN papers leads to images that are subjectively much worse than those presented in those papers (see Figure 24, left). We believe this may be due to our omission of WeightNorm, which the BadGAN authors have suggested plays an important role in their model's effectiveness [Dai (2008)]. We have had difficulty getting WeightNorm to work in our TensorFlow implementation, however.

We find that the best way to produce higher-quality images is to additionally feature

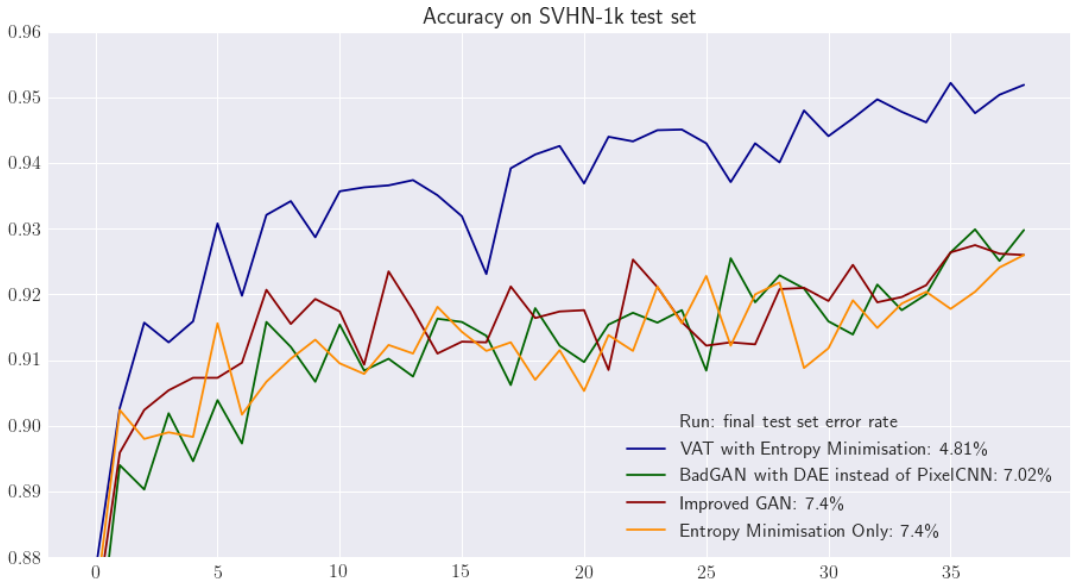


Figure 25: Performance of various model specifications on the SVHN-1k dataset.

match on earlier layers of the classifier (Figure 24, right). Specifically, we added a feature matching cost to the final layer of each of the three groups of convolutional layers described in the previous section. This also improved the diversity of the generated images; we found that other approaches to improving image quality tended to lead to mode collapse.[11]

Using just entropy minimisation as the cost for our unlabeled data, we achieve a test set error rate of 7.4% (Figure 25). This error is already lower than the best result reported in the original Improved GAN paper of 8.11%, using the same classifier architecture. Adding the Improved GAN costs to this model, using three-layer feature matching as described in the previous paragraph, netted us the exact same error rate.

We convert our Improved GAN into a BadGAN by adding variance matching[12] as a proxy for the entropy of the generated examples $H(p_G)$, instead of the pull-away term, as we found this to be faster, and perform just as well. Additionally we use our DAE proxy for the $p(\mathbf{x})$ threshold and gradient. This BadGAN-like approach further improved the test set error to 7.0%. Looking at Figure 25 however, it is clear that these improvements are too small to be deemed significant, particularly as, due to time constraints, we have only been able to average each model's performance over two runs here.

VAT with entropy minimisation, on the other hand, is the clear frontrunner in Figure 25, achieving a final test set error of 4.81%. This is substantially better than that reported in the original VAT paper of 6.83%, which is surprising as our implementation is based off the VAT authors' codebase. We put the difference down to running our version for more epochs, and possibly some small differences in the classifier architecture, which we re-implemented (as we were using the 'Conv-Small' architecture not included in the VAT codebase).

[11] Mode collapse is a common problem in GANs whereby only a small number of the modes of the true data distribution receive representation in the generative distribution.

[12] Variance matching is like feature matching, but we instead encourage the *variance* of each dimension of the generated features matches those of the true data. We have found this also increases diversity of the generated samples.

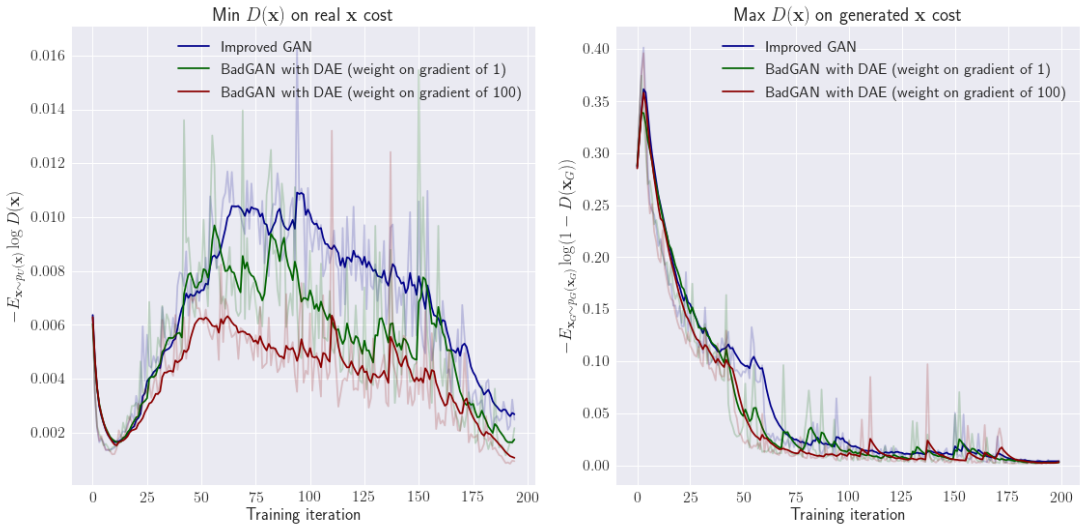


Figure 26: Impact of including a DAE as a means of ensuring the generator does not produce images above a certain $p(\mathbf{x})$ threshold (opaque lines are exponentially-smoothed with $\alpha = 0.2$).

8.2.3 Potential for replacing PixelCNN++ model with a DAE or discriminator from a GAN

In response to RQ7.2, we do find some evidence that our DAE approach may still work to replace the PixelCNN++ (were we in a situation where the generated images were actually contributing to classifier performance as intended). Specifically we find, similarly to PI-MNIST-100, that the classifier's $D(\mathbf{x})$ costs are lower for both the real and generated images when we enable the gradients flowing from the DAE model's estimate of $p(\mathbf{x})$ (Figure 26). This suggests that again the DAE gradient is reducing the quality of the generated images. We also can see (Figure 27) that the DAE gradients reduce the subjective quality of the generated images. It is of concern though, that the DAE-influenced images also appear to suffer from a higher degree of mode collapse.

Due to the resources required to estimate a PixelCNN++ on SVHN, we were not able

Figure 27: Images generated by our Improved GAN model on SVHN (left), versus those generated using our BadGAN model with DAE (right). Note that the BadGAN images are of subjectively poorer quality, but also appear to be suffering from mode collapse.

to estimate and compare this model's performance against that of a DAE on SVHN-1k. Regardless, without a clear benefit from the Improved GAN-style model over a baseline with just entropy minimisation and categorical cross-entropy loss (Figure 25), we are doubtful a PixelCNN++ model would provide much help in this instance. We first need to find some clear benefit from the Improved GAN approach, which, despite substantial efforts, we have been unable to achieve.

8.2.4 Hypotheses as to why our BadGAN implementation does not perform well on SVHN-1k

There are essentially three possibilities as to why our generated images are not regularising the classifier:

1. The images are 'too good' – that is, the generative distribution overlaps too much with the true data distribution, to the point that any beneficial regularisation is cancelled out by reducing model confidence in high-density regions.
2. The images are 'not good enough' – in this case the generated images are so far from the true data manifold that our model making less-certain predictions on them does not make any difference for its predictions on real data.
3. The classifier's goal, low-density separation, is already being achieved through the entropy minimisation cost.

Inspection of our model leads us to believe that the true cause is a combination of the first and third possibilities. We do see evidence that the generated images are 'good,' because we know the model is having some difficulty distinguishing between generated and real images (Figure 26). We also found that the model's categorical cross-entropy loss on the labeled examples goes up in the Improved GAN and BadGAN cases. This indicates that the classifier is diverting its modelling capacity towards being able to distinguish between real and generated images. This gap in cross-entropy cost between the entropy minimisation-only and Improved GAN models is not closed until the very end of our training run, so perhaps the Improved GAN model just needs a longer training period before it can start benefiting from the generated images. We did try this however, and found that longer training did not lead to any significant jump in performance.

We also believe that low-density separation is already being achieved by the supervised loss and entropy minimisation alone. This view is based on t-SNE [Maaten and Hinton (2008)] plots of the test set in the feature space, which already appears quite well-separated by regions of low density. Given we have 1000 labels (instead of just 100 in PI-MNIST-100), perhaps separating the ten classes is simply an easier task in this instance.

Given the difficulties faced in training our implementation of BadGAN on the SVHN-1k dataset, we leave addressing aspects of research questions RQ7.3 and RQ7.4 on this specific dataset to future work. Similarly, we aimed to extend our analysis to the CIFAR10-4k dataset. In our preliminary analysis however, we found that the issues faced with the SVHN-1k dataset were also faced in the CIFAR10-4k context, and as such we also leave analysis on this dataset to future work.

8.2.5 Experiments summary

The experiments described in Section 8 provided insight into our research questions (Section 7). Regarding RQ7.1, our experiments on PI-MNIST-100 gave clear support to the Improved GAN approach to BadGAN-style learning. Our findings there also supported the hypothesis that the CatGAN and Reverse KL approaches cause the model to 'forget' its class preferences in outer cluster regions too quickly. Last, we also found some evidence that the Reverse KL formulation of entropy may be more effective than using entropy itself. Thus we recommend that in future, if fitting models with cost functions that rely on entropy in some way, using the form $\sum_{k=1}^{K} \log p_k$ may perform better than $-\sum_{k=1}^{K} p_k \log p_k$.

Due to its resource intensity, the use of a PixelCNN++ model by BadGAN is one of its major drawbacks. Both our SVHN-1k and PI-MNIST-100 experiments suggest that our proposed approach of using a DAE instead of the PixelCNN++ model has merit. Although we were not able to replicate the results reported in Dai et al. (2017), our analyses suggest that the effect of the DAE and PixelCNN++ model are similar. When Dai et al. (2017) release code capable of replicating their results, it would certainly be of interest to substitute in our DAE approach, to see if it can achieve the same performance.

Regarding the orthogonality of the VAT and BadGAN approaches, our experiments on MNIST showed that these two methods are perhaps not as complementary as their authors suggest. As stated, we believe VAT and BadGAN are in fact quite similar, as both encourage low-density separation. Given this, we are more inclined to view these two approaches as substitutes, rather than complements.

Last, in response to RQ7.3, we found evidence through our experiments on PI-MNIST-100 that the generated images do indeed contribute to the learning of higher-order features in the classifier. We can thus recommend against any work in the direction of directly optimising characteristics of the feature space without backpropagating these objectives to earlier layers in the network.

9 Conclusions, practical recommendations and suggestions for future work

Our literature review showed that deep learning has changed the SSL landscape. We expect this trend to continue, as the flexibility of neural networks continues to facilitate new and creative approaches to SSL. We also believe however, that for the practical applicability of deep SSL research to improve, a stronger emphasis on larger, more challenging datasets is needed.

In Appendix A, we examined InfoReg, a method dating to 2006, and made derivations and updates making it ready for use in neural networks. Though this work is somewhat isolated from the core focus of this thesis, we believe it deserves mention here, as it exhibited competitive performance, and is derived from a sound theoretical base. Further improvements to this method may be a promising area of future research.

Through experiments on synthetic datasets, we demonstrated how some less-recent approaches – namely, entropy minimisation – have become more relevant in the era of deep learning. This owes to their suitability for use in neural network loss functions optimised via stochastic gradient descent. We also found that there are many functionals that can serve as proxies for entropy-related objectives. If it continues to be an integral part of many deep SSL models, then experimenting with alternative forms of entropy, namely Rényi entropies (which generalise a range of entropy functions) would be a beneficial area of future research.

Driven by these insights into the many entropy alternatives, one of our main research questions looked to determine which formulation was most effective for use in a BadGAN-style model. We experimented with various loss function formulations and found consistent theoretical and empirical evidence that the Improved GAN formulation is the most effective approach to BadGAN-style learning. It is less clear however, whether this conclusion can be transferred to other types of models that rely on the use of entropy more generally, or if it is only relevant in the BadGAN context.

Another key research question concerned whether the resource burden of the BadGAN model could be reduced by substituting the PixelCNN++ model it relies upon with a DAE. We did find encouraging supporting evidence for this idea on the MNIST dataset. Experiments on SVHN also provided some support. At this stage however, as we have not been able to replicate all of the original approach's results, further experimentation is needed to definitively confirm whether our method retains all of the benefits of the BadGAN model.

At a broader level, we have formed the view that the BadGAN model is not a particularly robust approach to SSL. In theory, and in our synthetic experiments, the basic idea works well, by allowing us to directly enforce low-density separation. However we see that as we move to more complicated datasets, it becomes increasingly difficult to generate images in the 'sweet spot' of the feature space, where they are not too realistic, nor too unrealistic, such that they provide useful information that regularises the classifier.

The dynamics of the GAN training process are already volatile. Adding additional large models, like a PixelCNN++, to try and temper this training process only increases this volatility. Given this, it is likely the case that for BadGAN to work in more complex scenarios, extensive hyperparameter tuning will be necessary. For this reason, we believe BadGAN will not scale well, and in light of this, we view the BadGAN and Improved GAN approaches as not particularly promising avenues for future SSL research.

Another research question focused on whether VAT and BadGAN are truly "orthogonal" [Dai et al. (2017)]. We have argued that VAT and BadGAN are, contrary to their author's assertions, actually quite similar, in that both encourage low-density separation. Perhaps Improved GAN and BadGAN add an ability to 'sharpen' the classifier's features which VAT does not provide. However, we find VAT to be robust and to require substantially less hyperparameter tuning. Thus, given their similarity we actually view VAT as a substitute for BadGAN, and a much more promising one at that.

Still, we believe other SSL methods possess attributes which VAT does not. VAT does

not explicitly contribute to the learning of semantically-meaningful features. Methods that add an unsupervised task to explicitly learn useful features (e.g. in Doersch et al. (2015), where a neural network learns to estimate the relative positionings of image patches) may complement VAT well. In addition, leveraging the effectiveness of Sajjadi et al. (2016)'s method in a combination with VAT may also prove effective, although this will depend on the degree to which VAT already promotes invariance to input and model perturbations.

GANs may still play a role in SSL's future. We believe however that the most promising paths for this are more likely to be via approaches 1 or 3 that we outlined in Section 4.3.2. These were:

- Methods which use the GAN training process as an unsupervised task to learn semantically-meaningful features, and;
- Methods which learn an inference model (i.e. an encoder network).

In our view, these directions seem more likely to lead to improvements in SSL performance through the use of GANs, particularly if they are optimised simultaneously with the classifier's cross-entropy loss. The BadGAN approach, in our view, relies upon learning a generator with a narrow definition that is difficult to achieve in practice, and its objective of low-density separation can be more reliably achieved through alternative approaches.

Part IV
Appendices

A Information regularisation for neural networks

We focus here on information regularisation, or InfoReg, a regularisation penalty from Corduneanu and Jaakkola (2006) that we gave an overview of in Section 4.2.2. InfoReg is a regularisation penalty that can be used on unlabeled data passed through a classifier. It encourages low-density separation by essentially penalising the data density at the decision boundary.

In this appendix, we derive a form of this regulariser that makes it ready for use in neural networks. We then evaluate its performance on our synthetic datasets and on the PI-MNIST-100 dataset. We find it to be competitive with current state-of-the-art methods in both of these contexts.

A.1 Derivation

We now show how InfoReg can be used to regularise a neural network classifier. Recall the expression for the InfoReg regulariser first presented in Section 4.2.2:

$$J(p) = \int p(\mathbf{x})\mathbf{tr}(F(\mathbf{x}))d\mathbf{x} \tag{A.1}$$

Where $F(\mathbf{x})$ is the Fisher Information [Ly et al. (2017)], given as:

$$F(\mathbf{x}) = \mathbf{E}_{C(k|\mathbf{x})}\left[(\nabla_{\mathbf{x}} \log C(k|\mathbf{x})) \cdot (\nabla_{\mathbf{x}} \log C(k|\mathbf{x}))^T\right] \tag{A.2}$$

Our contribution here is an empirical estimate for the InfoReg regulariser that can easily be added as a cost to a neural network, derived as:

$$\mathcal{L}_{\text{inforeg}} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{tr}\left(\sum_{k=1}^{K} C(k|\mathbf{x}_i)\left[(\nabla_{\mathbf{x}_i} \log C(k|\mathbf{x}_i)) \cdot (\nabla_{\mathbf{x}_i} \log C(k|\mathbf{x}_i))^T\right]\right) \tag{A.3}$$

$$= \mathbb{E}_{\mathbf{x}\sim p_U(\mathbf{x})}\left[\sum_{k=1}^{K} C(k|\mathbf{x}) \left\|\nabla_{\mathbf{x}} \log C(k|\mathbf{x})\right\|^2\right] \tag{A.4}$$

In the case of a neural network with a softmax output layer, we can further simplify this as:

$$= \mathbb{E}_{\mathbf{x}\sim p_U(\mathbf{x})}\left[\sum_{k=1}^{K} C(k|\mathbf{x}) \sum_{i=1}^{D}\left[\frac{\partial \log C(k|\mathbf{x})}{\partial \mathbf{x}_i}\right]^2\right] \tag{A.5}$$

$$= \mathbb{E}_{\mathbf{x}\sim p_U(\mathbf{x})}\left[\sum_{k=1}^{K} C(k|\mathbf{x}) \sum_{i=1}^{D}\left[\frac{\partial \log C(k|\mathbf{x})}{\partial C(k|\mathbf{x})}\frac{\partial C(k|\mathbf{x})}{\partial \mathbf{x}_i}\right]^2\right] \tag{A.6}$$

$$= \mathbb{E}_{\mathbf{x}\sim p_U(\mathbf{x})}\left[\sum_{k=1}^{K} C(k|\mathbf{x}) \sum_{i=1}^{D}\left[\frac{1}{C(k|\mathbf{x})}\frac{\partial C(k|\mathbf{x})}{\partial \mathbf{x}_i}\right]^2\right] \tag{A.7}$$

$$= \mathbb{E}_{\mathbf{x}\sim p_U(\mathbf{x})}\left[\sum_{k=1}^{K} \frac{1}{C(k|\mathbf{x})} \left\|\nabla_{\mathbf{x}} C(k|\mathbf{x})\right\|_2\right] \tag{A.8}$$

Where D is the dimensionality of the input $\mathbf{x}$ vector.

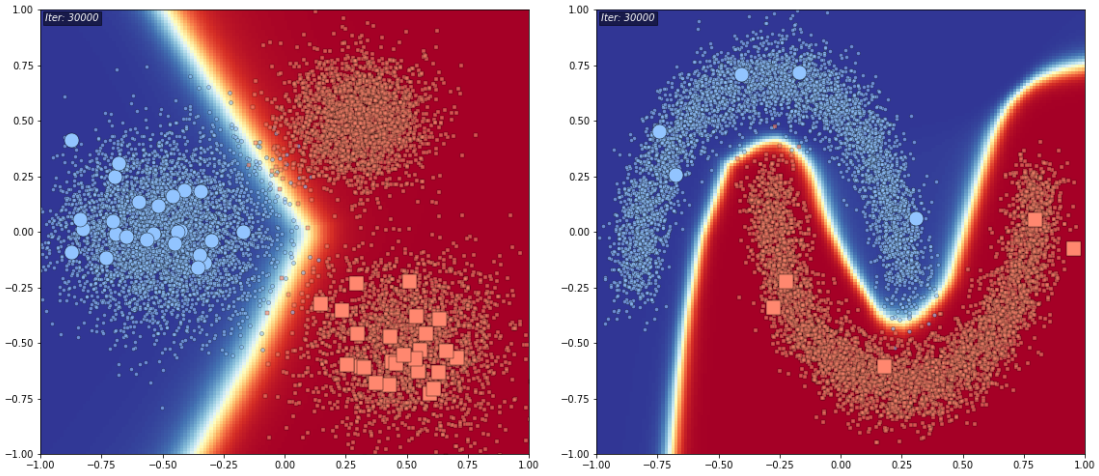


Figure 28: The result of training our neural network on the *Guassian Mixture* and *Moons* datasets using InfoReg (with a weight of 0.1), in addition to the RIM loss function described in Equation 6.3 for 30,000 training steps.

A.2 Intuition and experiments on synthetic datasets

The form of InfoReg in Equation A.8 provides for a more intuitive interpretation of this regulariser. We can see that the InfoReg loss term penalises model functions where the gradient of the output probability distribution with respect to the input $\mathbf{x}$ is large. This encourages smoothness by penalising the model's sensitivity to small changes in the input. We can also see that the penalty will apply more heavily in high-density regions. As such, the model is encouraged to not change its decision in high-density regions. Combining this with entropy minimisation in those regions should then lead to low-density separation, as the model is encouraged to be confident, and to not deviate from this confidence, in high-density regions. The penalty for class k is also scaled by $\frac{1}{C(k|\mathbf{x})}$: the inverse probability of that class. This ensures the scale of the penalty is invariant to the magnitude of the probability.

The effect of this penalty is that label information propagates to nearby points, as the model is encouraged to make confident decisions on these points (through cross-entropy), and to not change that decision in nearby regions. This propagation can be sustained over longer distances through entropy minimisation (Equation 6.1), which would encourage these nearby points to retain a high level of output confidence. This propagation process continues until it reaches a label of a different class, or a low-density region of the input space. As a result, the decision boundaries are pushed out towards low-density regions.

Through this mechanism of label propagation, we expect InfoReg, especially when coupled with entropy minimisation, to encourage low-density separation. Our synthetic experiments (Figure 28) suggest that this is indeed the case. We can see that the correct decision boundary for the Moons dataset is now learned. Meanwhile, the learned decision boundary on the Gaussian Mixture dataset is perhaps also more optimal, as it passes through less dense regions of the data. We should however note that without the average entropy maximisation component in Equation 6.3, the top-right cluster was not correctly classified. On the other hand, disabling entropy maximisation on the Moons dataset led to much more stable convergence.

A.3 FastInfoReg: overcoming InfoReg's speed issues

One issue with InfoReg is that it can be quite slow to compute the gradient $\nabla_{\mathbf{x}} \log C(k|\mathbf{x})$ for every k. Taking derivatives of a vector $\mathbf{y}$ with respect to another vector $\mathbf{x}$ is not currently fully supported in Tensorflow; it automatically sums over the gradients of each element when given a vector $\mathbf{y}$. As such, to implement InfoReg we needed to slice $\mathbf{y}$ into its scalar components and take the gradient of each with respect to $\mathbf{x}$ in a loop. Hence, training with Inforeg, which is rather slow, could presumably be sped up substantially if there was better support for calculating full Jacobians in Tensorflow.[13]

To try and get around this limitation, other, simpler formulations of InfoReg, still encapsulating the core idea of penalising large gradients of the output with respect to the inputs, were experimented with. Formulations assessed included penalising the gradient of the squared entropy with respect to $\mathbf{x}$, using only the gradient from the class k whose $C(k|\mathbf{x})$ was largest, and dropping the $C(k|\mathbf{x})$ before the gradient norm altogether. However, it appears to be crucial to weight the per-class gradient components by the probability of that class as predicted by the model.

A key breakthrough in overcoming these speed issues was the realisation that the sum over K in Equation A.4 is equivalent to taking an expectation over $C(k|\mathbf{x})$. As such, we could avoid calculating the full Jacobian for each training example by instead just sampling a single k and then only calculating the gradient for that particular dimension.

There were two approaches to this sampling that could be taken here. The first was to uniformly sample k from $\{1, \ldots, K\}$ and then multiply the resulting gradient by $C(k|\mathbf{x})$. The second was to sample from this set according to the multinomial distribution defined by $C(\cdot|\mathbf{x})$ and then not multiply the resulting gradient by this probability. We found the first approach to work far better, presumably because the multinomial sampling approach caused classes with low $C(k|\mathbf{x})$ to receive insufficient sampling attention. We call this approach 'FastInfoReg.'

A.4 Performance on PI-MNIST-100

After seeing the encouraging results on our synthetic datasets, we looked to evaluate FastInfoReg on the PI-MNIST-100 benchmark. We compare this to the performance of the Improved GAN model, which was one of the best performers from our main experiments (Section 8). As discussed in Section 4.2.2, Virtual Adversarial Training (VAT) [Miyato et al. (2017)] can also be viewed as quite similar to InfoReg. As such we also compare FastInfoReg's performance with that of VAT.

FastInfoReg has one main hyperparameter, which is the weight on the InfoReg cost. We found setting this to 3.0 worked best on the PI-MNIST-100 validation set. We also found it was quite important to tune the entropy minimisation cost; a weight of 0.55 was found to perform best on the validation set. We then ran this best configuration with six different random seeds. The evolution of test set performance is presented in Figure 29.

[13] We did not give up on this problem easily, however the speed of calculating Jacobians in Tensorflow is a documented and ongoing issue (see https://github.com/tensorflow/tensorflow/issues/675 for possible solutions that did not help in our case).

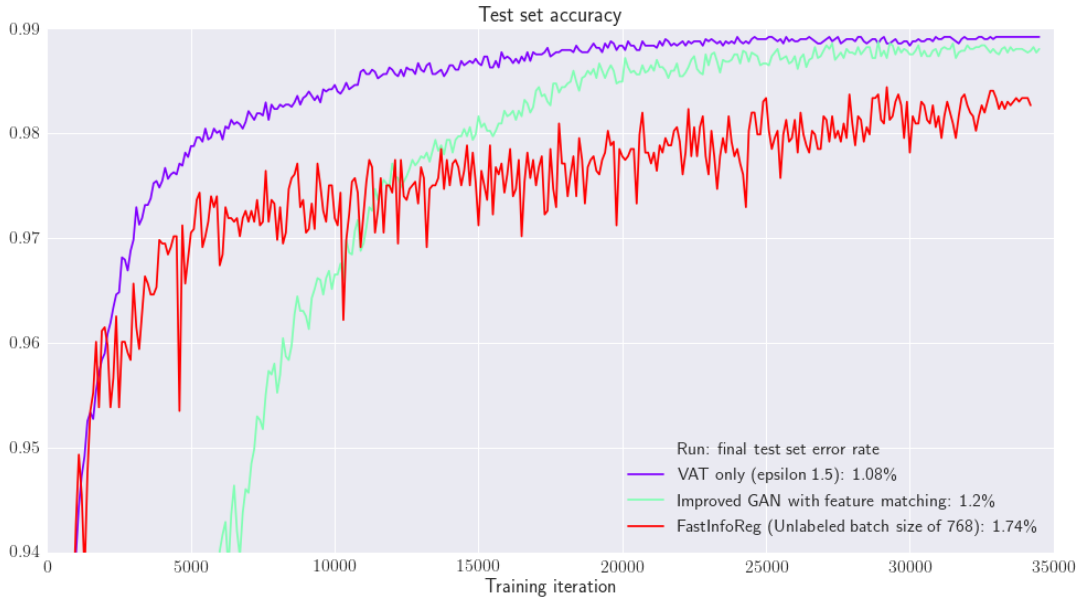


Figure 29: Test set accuracy during training of VAT-only, Improved GAN with feature matching, and FastInfoReg models. Note that while FastInfoReg was trained for the same number of steps, the size of its unlabeled batch optimised in each step was triple that of the other methods, at 768 examples.

We find InfoReg to be competitive with the other methods on this benchmark. On average, it achieved a final test set error rate of 1.74%, compared with 1.08% for VAT and 1.20% for the Improved GAN model. We should note however that due to the sampling process, the FastInfoReg test set error tends to be much noisier over time. To combat this, we tripled the unlabeled batch size to 768 to ensure the gradient sampling process is averaged over more examples, and thus would have lower variance. To keep the number of training steps the same, we also tripled the number of epochs (number of times we pass through the entire unlabeled training set).

Despite these increases in batch size and the number of training epochs, we find at these settings FastInfoReg is almost exactly as burdensome to run on a single NVIDIA Geforce TITAN X GPU, with VAT taking 42 minutes to complete a run and FastInfoReg taking 39 minutes. We also found at this batch size, the memory burden of both methods on the GPU was identical; presumably VAT also consumes more memory for its additional forward and backward passes of the perturbed input. To summarise, despite the fact that VAT still achieves better performance, and does so with lower variance, we believe our results are still noteworthy, particularly considering the fact that the InfoReg technique dates back to 2006. Taking steps to improve the performance of InfoReg may be a promising future research direction. Increased batch size or memory requirements, brought about due to the variance in the FastInfoReg gradient, could also be reduced by just keeping a running mean of the gradient updates and updating less frequently.

B Viewing entropy minimisation as a KL divergence minimisation problem

In Section 6.2, we gained some insights and additional loss functions by viewing our entropy maximisation goals as KL divergence minimisation problems. We can also apply this KL divergence interpretation when our objective is entropy *minmisation*. We do this by defining a theoretical entropy-minimising target probability distribution $r(\cdot)$ as one which takes on the argmax of our classifier's current output:

$$r(k) = \begin{cases} 1 & \text{if } k = \underset{k}{\text{argmax}}\ C(k|\mathbf{x}) \\ 0 & \text{otherwise} \end{cases}$$

The KL divergence between our classifier C (omitting its dependence on $\mathbf{x}$ for brevity) and this theoretical target distribution r is:

$$\begin{aligned} D\left(C\|r\right) &= \sum_{k=1}^{K} C(k) \log \frac{C(k)}{r(k)} \\ &= H\left(C\right) - \sum_{k=1}^{K} C(k) \log r(k) \end{aligned} \tag{B.1}$$

The term on the right hand side of this equation is the cross-entropy between distributions C and r, which we shall denote $H(C, r)$. Since, for classes other than $\underset{k}{\text{argmax}}\ C(k|\mathbf{x}, \boldsymbol{\theta})$, $\log r(k)$ will be $\log(0) = -\infty$, this will result in an infinitely large penalty for any C that is not equivalent to r. Since the output of the logistic function can never truly be 1.0, we can assume this penalty will always be $-\infty$ or ∞. Thus we view this penalty as not being usable for our purposes.

We can again reverse the KL divergence for our entropy-minimising target distribution:

$$\begin{aligned} D\left(r\|C\right) &= -\sum_{k=1}^{K} r(k) \log C(k) + \sum_{k=1}^{K} r(k) \log r(k) \\ &= \text{xent}(r, C) - H(r) \qquad \text{(B.2)} \\ &= \text{xent}(r, C) \qquad \text{(B.3)} \end{aligned}$$

Where the last line follows as $H(r) = 0$ by our definition. We can see that this particular means of encouraging entropy minimisation is equivalent to assigning a fake label to the current input, where the fake label is set to the most probable label as predicted by the current state of C. We then treat this instance as if it were a labeled data point (i.e., we backpropagate the cross-entropy loss). This approach is equivalent to minimising the Rényi entropy $H_\alpha(p) = \frac{\alpha}{1-\alpha} \log(||p||_\alpha)$ when α is defined as $\alpha \to \infty$ [Renner and Wolf (2004)].

This is also equivalent to Pseudolabel [Lee (2013)], a SSL approach described in Section 4.2.1. In that paper, the authors assert that this approach is the same as entropy minimisation. However, it would appear from the above derivations that this is not the case.

We can prove this difference more concretely by looking at the gradients flowing back to the weights connecting the penultimate layer of a neural network to its final layer (which is then converted to a probability distribution by applying the softmax function). If $\mathbf{f}$ were the activations at the penultimate layer, the gradients that would flow to the ith vector of weights $\mathbf{w}_i$ from this Reverse KL penalty would be:

$$\frac{\partial \text{xent}(r, C)}{\partial \mathbf{w}_i} = \begin{cases} (C_i - 1)\mathbf{f} & \text{if } i = \underset{k}{\text{argmax}}\ C(k|\mathbf{x}, \boldsymbol{\theta}) \\ C_i \mathbf{f} & \text{otherwise} \end{cases}$$

Here C_i represents the ith probability output of the classifier function. On the other hand, the gradients from an entropy minimisation (original KL) penalty to $\mathbf{w}_i$ would be:

$$\frac{\partial H(C)}{\partial \mathbf{w}_i} = -\mathbf{f} \cdot \left[(\log C_i + 1) C_i (1 - C_i) - \sum_{j \neq i}^{K} (\log C_j + 1) C_i C_j \right]$$

Thus, while these two penalties have the same goal (both are minimised when entropy is minimised), their path to this goal (via stochastic gradient descent) will be quite different, contrary to the assertion in Lee (2013) that these approaches are the same.

References

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. Software available from tensorflow.org.

Alain, G. and Bengio, Y. (2014). What regularized auto-encoders learn from the data-generating distribution. *The Journal of Machine Learning Research*, 15(1):3563–3593.

Arjovsky, M., Chintala, S., and Bottou, L. (2017). Wasserstein gan. *arXiv preprint arXiv:1701.07875*.

Bachman, P., Alsharif, O., and Precup, D. (2014). Learning with pseudo-ensembles. In *Advances in Neural Information Processing Systems*, pages 3365–3373.

Berthelot, D., Schumm, T., and Metz, L. (2017). Began: Boundary equilibrium generative adversarial networks. *arXiv preprint arXiv:1703.10717*.

Bishop, C. M. (1995). Training with noise is equivalent to tikhonov regularization. *Neural computation*, 7(1):108–116.

Blum, A. and Mitchell, T. (1998). Combining labeled and unlabeled data with co-training. In *Proceedings of the eleventh annual conference on Computational learning theory*, pages 92–100. ACM.

Chapelle, O., Scholkopf, B., and Zien, A. (2006). *Semi-Supervised Learning*. MIT Press.

Chapelle, O. and Zien, A. (2005). Semi-supervised classification by low density separation. In *AISTATS*, pages 57–64.

Coates, A., Ng, A., and Lee, H. (2011). An analysis of single-layer networks in unsupervised feature learning. In Gordon, G., Dunson, D., and Dudk, M., editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 215–223, Fort Lauderdale, FL, USA. PMLR.

Cohen, T. and Welling, M. (2016). Group equivariant convolutional networks. In *International Conference on Machine Learning*, pages 2990–2999.

Corduneanu, A. and Jaakkola, T. (2006). Data dependent regularization. *Semi-Supervised Learning, MIT Press.*, pages 163–182.

Curious AI Company, T. (2016). Learning by denoising part 2. connection between data distribution and denoising function. *https://thecuriousaicompany.com/connection-to-g/*.

Dai, Z. (2008). Correspondence via email, 5 august 2017.

Dai, Z., Yang, Z., Yang, F., Cohen, W. W., and Salakhutdinov, R. (2017). Good semi-supervised learning that requires a bad gan. *arXiv preprint arXiv:1705.09783*.

Denton, E., Gross, S., and Fergus, R. (2016). Semi-supervised learning with context-conditional generative adversarial networks. *arXiv preprint arXiv:1611.06430*.

Doersch, C., Gupta, A., and Efros, A. A. (2015). Unsupervised visual representation learning by context prediction. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1422–1430.

Donahue, J., Krähenbühl, P., and Darrell, T. (2016). Adversarial feature learning. *arXiv preprint arXiv:1605.09782*.

Dosovitskiy, A., Springenberg, J. T., Riedmiller, M., and Brox, T. (2014). Discriminative unsupervised feature learning with convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 766–774.

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

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*, pages 2672–2680.

Grandvalet, Y. and Bengio, Y. (2005). Semi-supervised learning by entropy minimization. In *Advances in neural information processing systems*, pages 529–536.

Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., and Courville, A. C. (2017). Improved training of wasserstein gans. *CoRR*, abs/1704.00028.

Guttenberg, N. (2016). Stability of generative adversarial networks. *Araya Research Blog*.

Haeusser, P., Mordvintsev, A., and Cremers, D. (2017). Learning by association-a versatile semi-supervised training method for neural networks. *arXiv preprint arXiv:1706.00909*.

He, K., Zhang, X., Ren, S., and Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *CoRR*, abs/1502.01852.

Hinton, G., Vinyals, O., and Dean, J. (2015). Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*.

Hinton, G. E. (2009). Deep belief networks. *Scholarpedia*, 4(5):5947.

International Data Corporation, T. (2014). The digital universe of opportunities: Rich data and the increasing value of the internet of things. *https://www.emc.com/leadership/digital-universe/2014iview/executive-summary.htm*.

Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456.

Joachims, T. (1999). Transductive inference for text classification using support vector machines. In *ICML*, volume 99, pages 200–209.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980.

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

Kodali, N., Abernethy, J. D., Hays, J., and Kira, Z. (2017). How to train your DRAGAN. *CoRR*, abs/1705.07215.

Krause, A., Perona, P., and Gomes, R. G. (2010). Discriminative clustering by regularized information maximization. In *Advances in neural information processing systems*, pages 775–783.

Krizhevsky, A. (2009). Learning multiple layers of features from tiny images.

Kumar, A., Sattigeri, P., and Fletcher, P. T. (2017). Improved semi-supervised learning with gans using manifold invariances. *arXiv preprint arXiv:1705.08850*.

Laine, S. and Aila, T. (2016). Temporal ensembling for semi-supervised learning. *arXiv preprint arXiv:1610.02242*.

LeCun, Y. (1998). The mnist database of handwritten digits. *http://yann.lecun.com/exdb/mnist/*.

LeCun, Y. (2016). Facebook post. *https://www.facebook.com/yann.lecun/posts/10153426023477143*.

Lee, D.-H. (2013). Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks.

Li, C., Xu, K., Zhu, J., and Zhang, B. (2017). Triple generative adversarial nets. *arXiv preprint arXiv:1703.02291*.

Ly, A., Marsman, M., Verhagen, J., Grasman, R., and Wagenmakers, E.-J. (2017). A tutorial on fisher information. *arXiv preprint arXiv:1705.01064*.

Maaløe, L., Sønderby, C. K., Sønderby, S. K., and Winther, O. (2016). Auxiliary deep generative models. *arXiv preprint arXiv:1602.05473*.

Maas, A. L., Hannun, A. Y., and Ng, A. Y. (2013). Rectifier nonlinearities improve neural network acoustic models. In *Proc. ICML*, volume 30.

Maaten, L. v. d. and Hinton, G. (2008). Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(Nov):2579–2605.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119.

Miyato, T., Maeda, S.-i., Koyama, M., and Ishii, S. (2017). Virtual adversarial training: a regularization method for supervised and semi-supervised learning. *arXiv preprint arXiv:1704.03976*.

Nash, J. (1951). Non-cooperative games. *Annals of mathematics*, pages 286–295.

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 *NIPS workshop on deep learning and unsupervised feature learning*, volume 2011, page 5.

Odena, A., Dumoulin, V., and Olah, C. (2016). Deconvolution and checkerboard artifacts. *Distill*, 1(10):e3.

Oord, A. v. d., Kalchbrenner, N., and Kavukcuoglu, K. (2016). Pixel recurrent neural networks. *arXiv preprint arXiv:1601.06759*.

Park, S., Park, J.-K., Shin, S.-J., and Moon, I.-C. (2017). Adversarial dropout for supervised and semi-supervised learning. *arXiv preprint arXiv:1707.03631*.

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

Pereyra, G., Tucker, G., Chorowski, J., Kaiser, Ł., and Hinton, G. (2017). Regularizing neural networks by penalizing confident output distributions. *arXiv preprint arXiv:1701.06548*.

Pezeshki, M., Fan, L., Brakel, P., Courville, A., and Bengio, Y. (2016). Deconstructing the ladder network architecture. In *International Conference on Machine Learning*, pages 2368–2376.

Radford, A., Metz, L., and Chintala, S. (2015). Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*.

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

Renner, R. and Wolf, S. (2004). Smooth rényi entropy and applications. In *Information Theory, 2004. ISIT 2004. Proceedings. International Symposium on*, page 233. IEEE.

Rifai, S., Dauphin, Y. N., Vincent, P., Bengio, Y., and Muller, X. (2011). The manifold tangent classifier. In *Advances in Neural Information Processing Systems*, pages 2294–2302.

Sajjadi, M., Javanmardi, M., and Tasdizen, T. (2016). Regularization with stochastic transformations and perturbations for deep semi-supervised learning. In *Advances in Neural Information Processing Systems*, pages 1163–1171.

Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A., and Chen, X. (2016). Improved techniques for training gans. In *Advances in Neural Information Processing Systems*, pages 2234–2242.

Salimans, T., Karpathy, A., Chen, X., and Kingma, D. P. (2017). Pixelcnn++: Improving the pixelcnn with discretized logistic mixture likelihood and other modifications. *arXiv preprint arXiv:1701.05517*.

Salimans, T. and Kingma, D. P. (2016). Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *Advances in Neural Information Processing Systems*, pages 901–909.

Springenberg, J. T. (2015). Unsupervised and semi-supervised learning with categorical generative adversarial networks. *arXiv preprint arXiv:1511.06390*.

Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *Journal of machine learning research*, 15(1):1929–1958.

Tompson, J., Goroshin, R., Jain, A., LeCun, Y., and Bregler, C. (2015). Efficient object localization using convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 648–656.

Ulyanov, D., Vedaldi, A., and Lempitsky, V. (2017). Adversarial generator-encoder networks. *arXiv preprint arXiv:1704.02304*.

Wang, R., Cully, A., Chang, H. J., and Demiris, Y. (2017). Magan: Margin adaptation for generative adversarial networks. *arXiv preprint arXiv:1704.03817*.

Zhang, X. and LeCun, Y. (2015). Universum prescription: Regularization using unlabeled data. *CoRR*, abs/1511.03719.

Zhao, J., Mathieu, M., and LeCun, Y. (2016). Energy-based generative adversarial network. *arXiv preprint arXiv:1609.03126*.

Zhu, X. (2007). Semi-supervised learning. *International Conference on Machine Learning (ICML) Tutorial*.

Zhu, X. and Ghahramani, Z. (2002). Learning from labeled and unlabeled data with label propagation.