Université de Montréal

Feedforward deep architectures for classification and synthesis

par David Warde-Farley

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

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

août, 2017

Résumé

Cette thèse par article présente plusieurs contributions au domaine de l'apprentissage de représentations profondes, avec des applications aux problèmes de classification et de synthèse d'images naturelles. Plus spécifiquement, cette thèse présente plusieurs nouvelles techniques pour la construction et l'entraînement de réseaux neuronaux profonds, ainsi qu'une étude empirique de la technique de «dropout», une des approches de régularisation les plus populaires des dernières années.

Le premier article présente une nouvelle fonction d'activation linéaire par morceau, appellée «maxout», qui permet à chaque unité cachée d'un réseau de neurones d'apprendre sa propre fonction d'activation convexe. Nous démontrons une performance améliorée sur plusieurs tâches d'évaluation du domaine de reconnaissance d'objets, et nous examinons empiriquement les sources de cette amélioration, y compris une meilleure synergie avec la méthode de régularisation «dropout» récemment proposée.

Le second article poursuit l'examen de la technique «dropout». Nous nous concentrons sur les réseaux avec fonctions d'activation rectifiées linéaires (ReLU) et répondons empiriquement à plusieurs questions concernant l'efficacité remarquable de «dropout» en tant que régularisateur, incluant les questions portant sur la méthode rapide de rééchelonnement au temps de l'évaluation et la moyenne géométrique que cette méthode approxime, l'interprétation d'ensemble comparée aux ensembles traditionnels, et l'importance d'employer des critères similaires au «bagging» pour l'optimisation.

Le troisième article s'intéresse à un problème pratique de l'application à l'échelle industrielle de réseaux neuronaux profonds au problème de reconnaissance d'objets avec plusieurs étiquettes, nommément l'amélioration de la capacité d'un modèle à discriminer entre des étiquettes fréquemment confondues. Nous résolvons le problème en employant la prédiction du réseau pour construire une partition de l'espace des étiquettes et ajoutons au réseau des sous-composantes dédiées à chaque sous-ensemble de la partition.

Finalement, le quatrième article s'attaque au problème de l'entraînement de modèles génératifs implicites sur des images naturelles en suivant le paradigme des réseaux génératifs adversariaux (GAN) récemment proposé. Nous présentons une procédure d'entraînement améliorée employant un auto-encodeur débruitant, entraîné dans un espace de caractéristiques abstrait appris par le discriminateur, pour guider le générateur à apprendre un encodage qui s'aligne de plus près aux données. Nous évaluons le modèle avec le score «Inception» récemment proposé.

Mots-clés: apprentissage de représentations profondes, apprentisage machine, réseau de neurones, apprentisage supervisé, apprentissage non-supervisé, dropout, fonction d'activation, réseau convolutionel, reconnaissance d'objets, synthèse d'images

Summary

This thesis by articles makes several contributions to the field of deep learning, with applications to both classification and synthesis of natural images. Specifically, we introduce several new techniques for the construction and training of deep feed-forward networks, and present an empirical investigation into dropout, one of the most popular regularization strategies of the last several years.

In the first article, we present a novel piece-wise linear parameterization of neural networks, *maxout*, which allows each hidden unit of a neural network to effectively learn its own convex activation function. We demonstrate improvements on several object recognition benchmarks, and empirically investigate the source of these improvements, including an improved synergy with the recently proposed dropout regularization method.

In the second article, we further interrogate the dropout algorithm in particular. Focusing on networks of the popular rectified linear units (ReLU), we empirically examine several questions regarding dropout's remarkable effectiveness as a regularizer, including questions surrounding the fast test-time rescaling trick and the geometric mean it approximates, interpretations as an ensemble as compared with traditional ensembles, and the importance of using a bagging-like criterion for optimization.

In the third article, we address a practical problem in industrial-scale application of deep networks for multi-label object recognition, namely improving an existing model's ability to discriminate between frequently confused classes. We accomplish this by using the network's own predictions to inform a partitioning of the label space, and augment the network with dedicated discriminative capacity addressing each of the partitions.

Finally, in the fourth article, we tackle the problem of fitting implicit generative models of open domain collections of natural images using the recently introduced Generative Adversarial Networks (GAN) paradigm. We introduce an augmented training procedure which employs a denoising autoencoder, trained in a high-level feature space learned by the discriminator, to guide the generator towards feature encodings which more closely resemble the data. We quantitatively evaluate our findings using the recently proposed Inception score.

Keywords: neural network, machine learning, deep learning, supervised learning, unsupervised learning, dropout, generative adversarial network, activation function, convolutional network, object recognition, image synthesis

Contents

List of Figures

List of Tables

List of Abbreviations

ALI	Adversarially Learned Inference
CNN	Convolutional Neural Network
DBM	Deep Boltzmann Machine
GAN	Generative Adversarial Network
GPU	Graphics Processing Unit
i.i.d.	Independent and Identically Distributed
ILSVRC	ImageNet Large-Scale Visual Recognition Challenge
KL	Kullback-Leibler (divergence)
LAPGAN	Laplacian Pyramid Generative Adversarial Network
mAP	Mean Average Precision
MLP	Multi-Layer Perceptron
MP-DBM	Multi-Prediction Deep Boltzmann Machine
NCE	Noise-Contrastive Estimation
PWL	Piecewise Linear
RGB	Red, Green, Blue
ReLU	Rectified Linear Unit
SGD	Stochastic Gradient Descent
SVHN	Street View House Numbers
SVM	Support Vector Machine
VAE	Variational Auto-Encoder
ZCA	Zero-phase Components Analysis

Acknowledgments

Much of the credit for my reaching this point is due to my mother, Joan Warde-Farley, and my father, the late Bernard Leo Farley.

My mother has been a constant source of support throughout this degree and the two preceding it, and guided us adeptly through the passing of my father in 2005. Among other traits, I have inherited her unmatched stubbornness: having surreptitiously overheard her expressing doubt in my seriousness about pursuing a PhD in Montreal, I knew for certain that I had to carry it forward. Her early incredulity of course gave way to a deluge of financial, logistical, emotional and moral support, as I always knew it would.

My father taught me a great deal during the two decades we shared, including the value of honesty, humility, perseverance, and clarity of purpose. I remain but an imperfect student of his ways. I trust that he would view the first doctorate on his side of the family as a compelling alternative to the career as a concert pianist that he once envisioned for me.

I would like to earnestly thank my doctoral advisor, Yoshua Bengio, for his encouragement, enthusiasm, patience, and guidance, for granting me the great pleasure of joining MILA (née LISA), and for being the organizing force thereof. It has been an immense honour to witness firsthand the lab's transformation into the deep learning juggernaut that it is today. I'd also like to extend a special thanks to Aaron Courville, with whom I interacted a great deal in the early days of my studies, and with whom I co-authored several of the articles presented herein, and to Vincent Dumoulin for his assistance in translating the summary of this thesis.

I have had the enormous fortune to benefit from many brilliant teachers and mentors even before my arrival in Montreal. In particular, I'd like to particularly thank Quaid Morris, my MSc supervisor, from whom I learned a great deal about being a scientist, and who encouraged me to pursue my interests even where they diverged from his own; the late Sam Roweis, whose contagious enthusiasm was matched only by the incredible depth and breadth of his scholarship, may he rest in peace; and Geoffrey Hinton, scientific renegade extraordinaire, who originally ignited my interest in machine learning and neural networks.

Part of this work was undertaken at Google in Mountain View, California. I'd like to thank everyone with whom I interacted during both of my internships, on the Brain team and the Image Understanding team respectively, and in particular my hosts, Rajat Monga and Drago Anguelov, as well as Andrew Rabinovich with whom I worked closely during the summer of 2014.

I am grateful to all the members of MILA, past and present, with whom I have interacted. Many of the lab's members became good friends outside of the context of the lab. I would in particular like to thank Ian Goodfellow, Guillaume Desjardins, Razvan Pascanu, Mehdi Mirza, Laurent Dinh, Vincent Dumoulin, Mathieu Germain, Li Yao, Yann Dauphin, Bart van Merriënboer and Yaroslav Ganin for their

extralaboratory camaraderie. I am delighted to be reunited with numerous MILA colleagues in my new position at DeepMind.

Certain individuals have gone beyond mere friendship and played pivotal roles in my years in Montreal, and may be unaware just how deeply certain small acts have shaped my life for the better. In that vein, I owe a particular debt of gratitude to each of James Bergstra and Dumitru Erhan.

All of the work in these pages is built upon open source scientific software, a noble cause to which I have done my best to contribute (sometimes at the expense of more pressing pursuits, in the finest of graduate school traditions). I would like to thank in particular all of the contributors to NumPy, SciPy, Matplotlib and IPython. I would especially like to thank the Theano core development team (notably Frédéric Bastien, Pascal Lamblin, Arnaud Bergeron) for all of their hard work on a tool that played an integral role in much of this research, as well as Ian Goodfellow, Vincent Dumoulin, Matt Grimes and others for their contributions to Pylearn2 alongside my own. It has also been a great pleasure to collaborate with Bart van Merriënboer, Dzmitry Bahdanau, Vincent Dumoulin, and Dmitriy Serdyuk on the Blocks and Fuel packages, which restored sanity to my research workflow.

I would like to graciously acknowledge financial support from Ubisoft, D-Wave Systems, the Canada Research Chairs, NSERC, CIFAR, and the Université de Montréal. I'd also like to thank Enthought Inc. for sponsoring my attendance of SciPy 2008 through 2012.

Last but not least, I'd like to thank my fiancée, Johanna, for her love and support, and Sheldon for his life-enriching affection and mischief.

1 Background

Machine learning is the study of artificial systems that can adapt or learn from data presented to them. In the seminal work of Valiant (1984), the definition of *learning* adopted is somewhat open-ended, but appropriate given the topics covered herein: "a program for performing a task has been acquired by *learning* if it has been acquired by any means other than explicit programming". Machine learning, in its various guises, has established itself as a near-ubiquitous tool in science and engineering, easing the development and deployment of automated systems for tasks where explicitly articulated "recipes" are difficult (or effectively impossible, *a priori*) to construct. Insofar as any agent considered intelligent ought to be able (and should frequently find it useful) to adapt its behaviour in light of observation and experience, the study of machine learning is an essential element in the pursuit of artificial intelligence.

The academic study of machine learning is commonly broken down into three main areas. *Supervised learning* deals with the discovery of input-output mappings for some task of interest given correct or approximately correct examples of said mapping. The setting in which a system predicts one of a fixed number of discrete *labels* given an input signal (such as predicting, from physiological measurements, the presence or absence of a disease) is commonly referred to as *classification*, whereas prediction involving well-ordered numerical targets (a company's stock price, for example) is known as *regression*. *Unsupervised learning* is an umbrella term for any procedure that operates on only "input", typically procedures that attempt to uncover some type of structure in the distribution of input signals. Prominent examples of unsupervised learning include *clustering*, where data points are grouped into one of many discrete groups; *decomposition* of a signal into (usually additive) parts, subject to some set of constraints (this can be thought of as a kind of "cause" discovery); and *density estimation*, whereby the learner attempts to identify a parameterization of the probability distribution from which the data is drawn. Note that these categories of unsupervised learning are not mutually

exclusive: many density estimation methods have a decomposition or clustering interpretation, and vice-versa. Finally, *reinforcement learning* concerns systems that implement a *policy* mapping sequences of stimuli (i.e. the state of the "world", as experienced by an autonomous agent) to actions; however, an examination of this paradigm lies beyond the scope of this thesis.

1.0.1 Parametric and non-parametric learning

Orthogonal to the question of whether a method is supervised or unsupervised is the distinction between parametric and nonparametric methods. Though the boundary is defined differently by some authors, we adopt the following convention: *parametric* methods can be characterized as those that represent a solution in terms of a finite set of numerical parameters; crucially, the size of this set remains fixed throughout learning and is independent of the amount of training data. By contrast, a *non-parametric* method is one in which the complexity of the hypothesized solution is adaptive to the complexity of the task or the amount of available training data. The canonical non-parametric method is a "nearest neighbour" classifier, where classification proceeds by comparing a test example to every example seen during training, and predicting the label corresponding to the training example most similar to the test example (e.g., in terms of Euclidean distance). "Learning" then corresponds to simply storing the training set. Non-parametric methods are powerful in that they can often perform impressively while making very few (or very broad) inductive assumptions, but this flexibility often comes at the cost of computational complexity in both space and time – in the case of naively implemented nearest neighbour classification, both the amount of memory or disk space required (to store the training set) and the amount of computation required to classify a new point scales linearly with the size of the training set.

The methods considered in the remainder of this thesis are parametric in the sense that any instance considered in isolation obeys our conventions for a parametric method, with a number of parameters determined *a priori* and remaining fixed during training (though the method described in chapter 7 invokes two phases of such training). However, one property which all of these methods share is that the number of learnable parameters that describe the data distribution is effectively a free parameter, and can always be made larger in response to the availability of

greater amounts of training data. Furthermore, it is common to optimize over possible sizes of the parameter set in an automated "outer loop" by considering many instances in the same family and choosing the one that performs best (according to some criterion) on data not used during training. The combined selection and learning procedure is thus effectively non-parametric, by means of exploring a family of parametric learners.

1.0.2 Parameters and hyperparameters

In the machine learning literature, the term *parameter* is typically reserved for quantities that are adapted during the course of learning. However, the vast majority of machine learning methods will have one or more *hyperparameters* that must be specified beforehand, such as the number of basis functions or latent variables, or the step size of a numerical optimization procedure. For many methods, correct selection of the relevant hyperparameters is crucial to obtaining good performance.

1.1 Formalizing learning

The learning task, whether supervised or unsupervised, can be formalized as follows: suppose the possible inputs to our machine lie in a domain $\mathcal{D}$ and are distributed throughout a space $\mathcal{S}$ such that $\mathcal{D} \subseteq \mathcal{S}$, according to a probability distribution $p_{\mathcal{D}}$. Given a hypothesis space $\mathcal{P}$ (i.e., the space which contains all possible settings of the learnable parameters), and a *loss function* $\mathcal{L} : \mathcal{P} \times \mathcal{D} \to \mathbb{R}$ that describes, in some way, the performance of the learning machine on a given data example, learning seeks to identify parameters $\theta^\star \in \mathcal{P}$ such that

$$\theta^\star = \arg\min_\theta \int_{\mathcal{D}} \mathcal{L}(\theta, \mathbf{v}) p_{\mathcal{D}}(\mathbf{v}) d\mathbf{v} \tag{1.1}$$

i.e., $\theta^\star$ minimizes the expected loss with respect to all valid inputs $\mathbf{v}$ to the learning machine. In the supervised case, $\mathcal{D}$ is the set of all corresponding input-output pairs $(\mathbf{x}, \mathbf{y})$, and an intuitive choice for the loss function in the case of classification, with the learner parameterizing a function f_θ that outputs a predicted label:

$$\mathcal{L}_{\text{sup}}(\theta, (\mathbf{x}, \mathbf{y})) = \begin{cases} 0, & \text{if } f_\theta(\mathbf{x}) = \mathbf{y} \\ 1, & \text{otherwise} \end{cases} \tag{1.2}$$

known as the *zero-one misclassification loss.*[i] In an unsupervised setting, targets are omitted and the choice of loss function may involve a task such as reconstructing the input from an encoded form, a denoising criterion (Vincent et al., 2008), or a likelihood term deriving from a probabilistic model.

It is quite common in both supervised and unsupervised settings to adopt as the loss function the negative of the logarithm of a (parameterized) probability density, that either captures the *conditional distribution* of the desired outputs given the inputs (in the supervised setting) or the probability distribution of the inputs themselves in the (usually high-dimensional) space in which they are embedded (in the unsupervised setting). For instance, *logistic regression* is a supervised classification method which models the conditional distribution of targets in $\{0, 1\}$ as a monotonic function of a linear combination of inputs $\mathbf{x} \in \mathbb{R}^d$,

$$p_\theta(\mathbf{y}|\mathbf{x}) = \sigma(\mathbf{w}^\mathsf{T}\mathbf{x} + b)^y (1 - \sigma(\mathbf{w}^\mathsf{T}\mathbf{x} + b))^{(1-y)} \tag{1.3}$$

where $\theta = \{\mathbf{w}, b\}$ are adjustable parameters, and $\sigma : \mathbb{R} \to [0, 1]$ is the *logistic sigmoid function*:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{1.4}$$

The loss function corresponding to logistic regression for $(\mathbf{x}, y)$ for $y \in \{0, 1\}$ is then given by

$$-\log p_\theta(y|\mathbf{x}) = -y \log \sigma(\mathbf{w}^\mathsf{T}\mathbf{x} + b) - (1 - y) \log(1 - \sigma(\mathbf{w}^\mathsf{T}\mathbf{x} + b)) \tag{1.5}$$

More generally, if θ parameterizes a probability model p_θ such that $p_\theta(\mathbf{x}) > 0$ for every $\mathbf{x} \in \mathcal{D}$, and we define $\mathcal{L}(\theta, \mathbf{x}) = -\log p_\theta(\mathbf{x})$, then $\mathcal{L}$ is known as the *cross-*

i. As this loss is non-smooth, it is often desirable to use smoother proxies for the raw misclassification error.

entropy between the true data distribution $p_{\mathbf{D}}$ and the model distribution p_θ, which is closely related [i] to the *Kullback-Leibler divergence*, a commonly employed measure of the difference between two probability distributions (Kullback and Leibler, 1951).

In most settings, the minimization in (1.1) is impossible to perform exactly, as we only have access to a finite subset of an extremely large or possibly infinite $\mathcal{D}$. Instead, we must be content to optimize a proxy for this loss on a finite *training set* $\mathbf{v}^{(1)}, \mathbf{v}^{(2)}, \ldots, \mathbf{v}^{(N)}$. It is most often assumed that each point in this training set is sampled independently from the same distribution $p_{\mathcal{D}}$, and that the examples seen after learning is complete (at "test time") will be drawn according to the same distribution, i.e. that the data is *independent and identically distributed*, and thus the proxy loss most often chosen can be thought of as a simple Monte Carlo approximation to the expectation above:

$$\theta^\star \approxeq \hat{\theta}^\star = \arg\min_\theta \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(\theta, \mathbf{v}^{(i)}) \tag{1.6}$$

The assumption that each point in the training set is drawn *independently* from an *identical distribution* (i.i.d.) means that their joint probability of being drawn is, by the third of Kolmogorov's axioms of probability, merely the product of their corresponding marginal probabilities, i.e. $p_{\mathcal{D}}(\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(N)}) = \prod_{i=1}^{N} p_{\mathcal{D}}(\mathbf{x}^{(i)})$. Adopting the same assumption for a probabilistic model p_θ and taking the loss function as $\mathcal{L}(\theta, \mathbf{v}) = -\log p_\theta(\mathbf{v})$, minimizing the average empirical loss above is equivalent to maximizing the *joint probability* of the observations. The sum in (1.6) is commonly known as the *log likelihood* of the training set (with the corresponding product of probabilities being known simply as the *likelihood*). The optimization problem posed in (1.6) is thus known as *maximum likelihood* estimation [ii], and is perhaps the most popular and successful approach to machine learning. With the i.i.d. assumptions, it can be proven that maximum likelihood estimation is *consistent* if the training set is drawn i.i.d. from a distribution $p_\theta^\star$ within some parametric family of distributions $\mathcal{P}$. Then the distribution $p_{\hat{\theta}^\star}$ obtained by maximum likelihood estimation on a finite training set will converge, in terms of decreasing

i. i.e. equal up to an additive constant, the negative entropy of the true data distribution $H(p_{\mathcal{D}}) = \int_{\mathcal{D}} p_{\mathcal{D}}(\mathbf{x}) \log p_{\mathcal{D}}(\mathbf{x}) d\mathbf{x}$

ii. The $\frac{1}{N}$ is of course optional and does not change the solution, but is often useful to include, e.g. to compare across different sizes of training sets.

Kullback-Leibler divergence, to $p_\theta^\star$ as the amount of training data increases. The central concern of machine learning is that of favourable performance on data not encountered during training, i.e. of *generalizing* beyond the training set; in this light, consistency is certainly a desirable property.

Note that maximum likelihood, while popular, is far from unique in this respect. Other consistent approaches to parameter estimation have been explored, often expressly to address shortcomings of maximum likelihood in certain settings. Optimizing a lower bound on an intractable log likelihood (Saul and Jordan, 1996) is one popular technique, whereas other procedures do not optimize the log likelihood even in this indirect fashion (Hyvärinen, 2005; Gutmann and Hyvarinen, 2010). In chapter 8, we will introduce another parameter estimation procedure of the latter type.

Optimization of the model parameters with respect to the loss can be performed (presuming that the loss is smooth and differentiable almost everywhere) by any number of (usually gradient-based) numerical optimization techniques. Of particular import for the methods discussed here are methods based on *stochastic gradient descent*, a generalization of simple *steepest descent* minimization (which adjust the parameters in the direction of the negative gradient). The key idea behind stochastic gradient descent is that, as the term to be minimized in (1.6) is an expectation computed over the training set, the gradient can be approximated by an average over $N' << N$ terms in the sum, or even a single term. For very large datasets, stochastic gradient descent can allow learning to progress much more rapidly than so-called *batch* methods which compute the exact cost and its exact gradient by exhaustively summing over the training set. Stochastic gradient descent also admits the possibility of *online* learning in which data arrives in a continuous, possibly evolving stream.

1.2 Probabilistic graphical models

For machine learning procedures with probabilistic semantics, the language of *graphical models* has become a standard way of describing these semantics. Briefly, nodes (vertices) of a graph represent random variables and edges denote (possible)

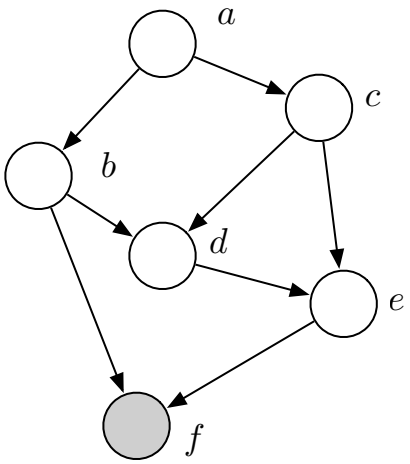

Figure 1.1 – A directed graphical model for the family of models whose joint probability distribution factorize as $p(a, b, c, d, e, f) = p(a)p(b|a)p(c|a)p(d|b, c)p(e|c, d)p(f|b, e)$. The variable f is observed.

conditional dependence relationships between them. By analogy with the conventional definition of independent random variables, two random variables a and b are said to be *conditionally independent* given a random variable c, if and only if

$$p(a, b|c) = p(a|c)p(b|c). \tag{1.7}$$

Note that two random variables can be *marginally* dependent and conditionally independent given a third observed random variable, and vice versa.

In printed form, shaded nodes in a graphical model typically represent observed quantities, while unshaded nodes represent unobserved or latent quantities. Most applications of graphical models in machine learning involve such *latent variables*, which may or may not have semantics corresponding to some physical reality. Such latent variables sometimes represent real but unobserved quantities, such as the true underlying quantity that has been measured and corrupted by a noisy sensor, or may otherwise more generally modulate or explain structured interactions between observed quantities.

1.2.1 Directed models and explaining away

Directed graphical models (also frequently known as *Bayesian networks* or *Bayes nets*) characterize a factorization of the joint probability density function into a product of normalized probability density (or mass, in the discrete case) functions, where a node x_i without parents (i.e., no incoming directed edges) contributes a marginal density $p(x_i)$, and a node x_i with parents $\mathbf{x}_{\pi(i)}$ contributes a conditional distribution $p(x_j|\pi(x_j))$, and so the joint distribution described by a *directed, acyclic graph* takes the form

$$p(x_1, x_2, \ldots, x_K) = \prod_{i=1}^{K} p(x_i|\pi(x_i)) \tag{1.8}$$

where $\pi(x_i)$ is the set of parents of node x_i, and we abuse notation to let $p(x_i|\{\}) = p(x_i)$. The graph semantics denote only dependence of one random variable on another and say nothing of the particular functional form, and thus parent-child relationships can be relatively arbitrary: in the case of a single discrete child node x_c with a single discrete parent x_p, one could imagine a 2-dimensional table T of values with rows enumerating states $\{d_1, d_2, \ldots, d_P\}$ of the parent and columns enumerating states $\{c_1, c_2, \ldots, c_M\}$ of the child, the values of the table representing the conditional probability of the child state given the parent state, i.e. $T_{ji} = P(x_c = c_j|x_p = d_i)$, with columns of the table summing to 1. More generally, if the child's probability density or mass function takes a specific parametric form, the value of a parent might serve as a parameter – for example a Beta-distributed parent might serve as the p (probability of "success", or heads in a coin flip) parameter for a Bernoulli-distributed child node. A discrete parent variable might index into a list (or lists) of parameters for a child variable. For example, a *mixture of Gaussians* can be written as a directed graphical model involving two nodes: a discrete random variable c with parameters $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_K)^\mathsf{T}$ such that $\sum_k \alpha_k = 1$,[i] and an observation vector $\mathbf{x}$ that is parameterized by K vectors in $\mathbb{R}^d$ $\mu_1, \mu_2, \ldots, \mu_K$, and K positive-definite matrices in $\mathbb{R}^{d\times d}$, $\Sigma_1, \Sigma_2, \ldots, \Sigma_K$. Given an (observed, or sampled) value c' for c (and treating the states of c as integers for notational convenience,

i. Technically we require only $K - 1$ parameters, since the last is fully determined by the sum constraint.

Figure 1.2 – A directed latent variable model. Each node may represent scalar or vector random variables; the rules of conditional independence described in Figure 1.3 ensure that the semantics are the same in either case, as long as there is a bipartite separation between observed and unobserved variables.

even though they are merely distinct states with no inherent order), then $\mathbf{x}$ is distributed as $\mathcal{N}(\mu_{c'}, \Sigma_{c'})$, the multivariate Gaussian distribution with mean $\mu_{c'}$ and covariance $\Sigma_{c'}$. Thus the realized value of the random variable c acts as an index into a list of mean parameters and a list of covariance parameters for its child node. The mixture of Gaussians is an instance of what we shall term a *directed latent variable model*, one of the most commonly studied structures in probabilistic machine learning, its general form depicted in Figure 1.2.

Conditional independence in directed models is easly described through the simple *Bayes ball algorithm* of (Shachter, 1998). The algorithm supposes a simulated ball to be bouncing from node to node on the graph; if the ball cannot reach one node x_i from another node x_j given the rules of the simulation, then the two are *conditionally independent* given the observed quantities. The rules are as follows: in all situations except one, the ball bounces off of observed (shaded) nodes (back in the direction it came) and passes through unobserved (unshaded) variables. The exception arises when two unobserved variables are jointly parents of a third variable (a *collider*); in this case, the rules are reversed: an *observed* third variable allows the ball to pass, whereas an *unobserved* third variable blocks it.

The situation described above, i.e. conditional coupling of two marginally independent parent random variables through an observed collider node, is known as *explaining away* (Pearl, 1988), and is an important concept in probabilistic reasoning. From a probabilistic modeling perspective, it is necessary for representing many realistic scenarios – an illuminated "check engine" light may mean that the car's engine needs to be serviced *or* that the fault sensor is misbehaving, but

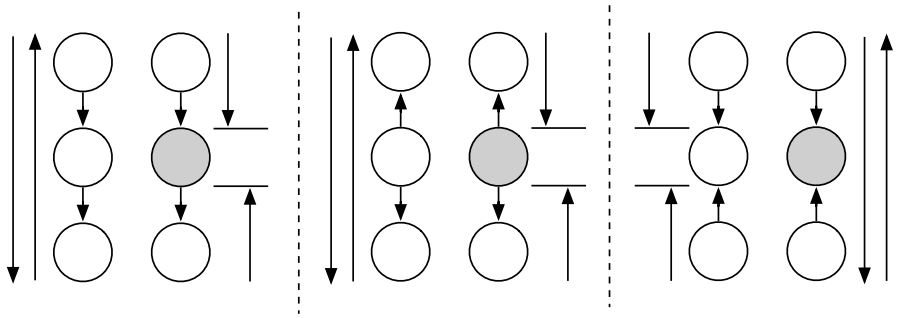

Figure 1.3 – An illustration of the Bayes-ball algorithm. Observed nodes block the flow of conditional dependence except in the case of "explaining away" on the far right.

given the observed evidence, these two causes offer *competing* explanations that are unlikely to both be true. From a computational perspective, explaining away (especially with discrete latent variables) often complicates *inference*, the characterization of the *posterior distribution* of unobserved variables given the observed variables, accomplished by means of *Bayes' rule*:

$$p(\mathbf{x}_{\text{latent}}|\mathbf{x}_{\text{observed}}) = \frac{p(\mathbf{x}_{\text{latent}})p(\mathbf{x}_{\text{observed}}|\mathbf{x}_{\text{latent}})}{p(\mathbf{x}_{\text{observed}})} \tag{1.9}$$

$$= \frac{p(\mathbf{x}_{\text{latent}})p(\mathbf{x}_{\text{observed}}|\mathbf{x}_{\text{latent}})}{\int_{\mathbf{x}_{\text{latent}}} p(\mathbf{x}_{\text{latent}})p(\mathbf{x}_{\text{observed}}|\mathbf{x}_{\text{latent}})} \tag{1.10}$$

Inference is a useful operation in its own right but also a critical component of several procedures for parameter estimation (Dempster et al., 1977). The difficulty arises when the denominator in (1.10) is intractable – for example, when the latent state is discrete and combinatorial. Intractable posterior distributions can be dealt with approximately, however, either by sampling or via deterministic *variational* approximations to the posterior distribution (Saul and Jordan, 1996).

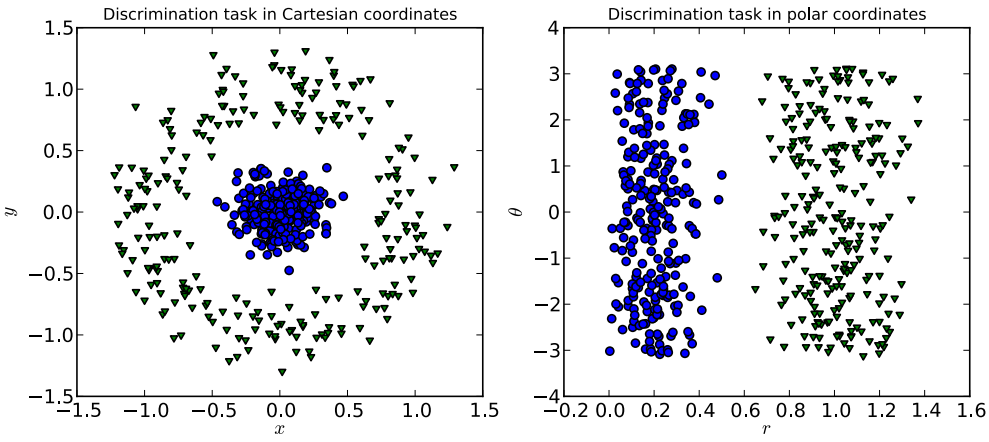

Figure 1.4 – Illustration of how changing representations can simplify a supervised learning problem. Here, a polar coordinates transform makes the problem linearly separable.

1.3 Neural Networks

1.3.1 Supervised learning

In supervised learning scenarios, specific representation of the input data employed can have a profound impact on successful generalization. Much of the work involved in practical applications of machine learning amounts to the manual design of *features*, deterministic functions of the raw input designed such that the off-the-shelf supervised learning algorithm can easily interrogate the structure of the problem. In particular, methods based on a linear combination of input features such as logistic regression or the linear support vector machine (Cortes and Vapnik, 1995) have the desirable property that their loss functions are *convex*: importantly, there is one global minimum, and its approximate location can be determined by a variety of techniques from the convex optimization literature. In many real-world scenarios, however, linear decision boundaries are insufficiently flexible: a linear classifier cannot even learn the exclusive-OR function, as no linear decision boundary can be drawn between the examples corresponding to the "true" (1) output and the "false" output. Another example is shown in Figure 1.4, whereby a simple change of variables (to polar coordinates) makes an otherwise more challenging classification problem linearly separable in the new space.

One relatively successful attempt to address this problem is via the "kernel

trick" (Aizerman et al., 1964): the solution to the optimization of the support vector machine's loss function is expressable in terms of inner products between training examples (Boser et al., 1992), and these inner products can be replaced with any *kernel function* satisfying mild conditions while retaining the convexity of the loss function. Kernel functions can be chosen such that they correspond to inner products in higher-dimensional spaces, or even infinite-dimensional Hilbert space, in which there exists a suitable linear decision boundary. It has been argued by Bengio and LeCun (2007) that the generalization capabilities of SVMs, especially when used in tandem with so-called "universal" kernels, is of a limited and *local* (in feature-space) character, and that richer mappings are necessary for nonlocal generalization. Kernelized SVMs are non-parametric in the sense that they depend on storing the *support vectors*, those training examples that lie on the margin adjacent to the decision boundary; for complicated classification problems this can lead to high computational complexity at test time, as well as a relatively large memory footprint.

Alternatively, multi-layer perceptrons (Rumelhart et al., 1986), also known as *feed-forward neural networks* offer a richer parametric approach, with one or more layers of nonlinear basis functions conventionally known as *hidden units* that are linearly combined in the output layer. For example, a multi-layer perceptron for binary classification with a single layer of hidden units could be parameterized as

$$h(\mathbf{x}) = s(V\mathbf{x} + \mathbf{c}) \tag{1.11}$$

$$o(\mathbf{x}) = \sigma(\mathbf{w}^\mathsf{T} h(\mathbf{x}) + b) \tag{1.12}$$

where V, $\mathbf{w}$, $\mathbf{c}$ and b are learnable parameters, and s is some elementwise nonlinearity. The output of the hidden units $h(\mathbf{x})$ is a learned, nonlinear transformation of the raw input, which can be adapted by gradient descent to the classification task at hand. Replacing $\sigma(\mathbf{w}^\mathsf{T}\mathbf{x}+b)$ with $o(\mathbf{x})$ in (1.3) and (1.5) gives us a more flexible extension of logistic regression that can be trained by gradient descent on the same loss function (1.5). Gradients on $\mathbf{w}$ and b are unchanged from standard logistic regression, while gradients on V and $\mathbf{c}$ (and, in general, the lower-layer parameters of any multilayer perceptron) are efficiently computable via the *backpropagation* algorithm (Rumelhart et al., 1986). Where simpler methods traditionally relied upon handcrafted *features* – fixed, hand-engineered transformations of the raw input –

neural networks offer the attractive possibility of learning to extract features using the hidden units of the network.

While such networks, even with a single layer of hidden units, are provably universal approximators (Hornik et al., 1989) (i.e., with enough hidden units they can approximate, to an arbitrary degree of accuracy, any continuous function on compact subsets of $\mathbb{R}^n$) this power comes at an additional cost. In gaining expressivity, the convexity of the loss function is sacrificed; identification of a global optimum of the loss function is no longer guaranteed. While a single layer of hidden units may be sufficient to approximate any input-output mapping arbitrarily well, this may come at the cost of representational (and hence statistical) efficiency (see Bengio and LeCun (2007) for detailed arguments to this effect). Multiple hidden layers can demonstrably lead to more efficient parameterizations (Bengio, 2009), but conventional numerical optimizers, until recently, notoriously failed to effectively train networks with more than one or two hidden layers (Glorot and Bengio, 2010).

The recent wave of success in training deep neural networks is attributable to several factors. Recent work has yielded a better understanding of factors such as initialization and gradient acceleration methods (Sutskever et al., 2013) which play a crucial role in optimization. Recent years have seen the replacement of sigmoidal nonlinearities (namely the logistic and hyperbolic tangent functions) with non-saturating nonlinearities; Jarrett et al. (2009) first investigated several rectification nonlinearities in convolutional object recognition architectures, focusing on the absolute value rectification, while Nair and Hinton (2010) showed that the half-wave rectifier $\max(0, x)$, which they dubbed the Rectified Linear Unit (ReLU), could be fruitfully applied within the context of restricted Boltzmann machines. Glorot et al. (2011) showed that rectified linear units could be used to train very deep multilayer perceptrons without need of sophisticated initializations based on unsupervised pre-training. Piecewise linear activation functions, including rectifiers, give rise to networks which are piecewise affine functions, through which gradient signal propagates much more readily (see chapter 3 for an extended exploration of this topic). Finally, the availability of large amounts of data, and the use of commodity graphics processing units for the rapid training of these computationally intensive networks (Raina et al., 2009) has allowed practitioners to identify regimes of high performance that were previously obscured by small sample sizes and prohibitive training times.

1.3.2 Encoding domain knowledge

We have thus far considered learning systems which operate on training cases which are arbitrary vectors in $\mathbb{R}^n$ and pay no heed to structured relationships between elements of these vectors. However, many signals of interest are highly structured, with elements having a natural topology in space or in time. A 32×32 pixel RGB image can be represented as a vector in $\mathbb{R}^{3072}$ and processed by any algorithm that is agnostic to the fact that it is dealing with pixels, but this neglects readily available domain knowledge about the structure of the problem. Just as machine learning practitioners often engineer discriminative features of the input based on domain knowledge, so can one infuse domain knowledge into learnable feature extraction systems.

One way of introducing domain knowledge in a neural network is by restricting the connectivity pattern, or receptive field, of individual hidden units. Another way is to force different hidden units, operating on different inputs, to share the same parameters – that is, to process different inputs in exactly the same fashion. Taken together, these strategies form the bedrock of the most successful class of neural networks to date.

Convolutional neural networks

In the case of images, it is reasonable to assume that primitive features useful for classification, such as edges or corners, will have a spatially local character. A layer of hidden units whose individual connectivity patterns tile the image with small, overlapping receptive fields will thus force the network to learn features which are localized in space; the early processing layers of the mammalian visual system are known to have a similar structure (Hubel and Wiesel, 1959).

The locally-connected regime described above would grant each locally connected hidden unit its own, independent weights. However, a useful property of images is that semantics are often preserved across translations in space; a bird is still a bird no matter if it appears perched on a window sill or on a tree branch. One can thus gain statistical efficiency by replicating *the same weights* across all spatial locations, allowing the network to detect a given, spatially localized pattern no matter where it appears. It is straightforward to show that the gradient of the surrogate loss with respect to the shared weights of these spatially replicated units

is simply the sum of the gradients with respect to each of its instantiations.

A spatially replicated linear transformation on local pixel neighbourhoods is precisely equivalent to a discrete, 2-dimensional convolution[i], a common primitive in image processing algorithms. *Convolutional neural networks* (LeCun, 1989; LeCun et al., 1998) incorporate these insights by replacing multiplication by weight matrices in neural networks with a linear transformation based on 2-dimensional convolutions. More precisely, some subset of a image's M input channels are convolved with 2-dimensional filters, and the results summed together; this is repeated N times, yielding N outputs from a total of $N \times M$ learnable 2-dimensional filters. For each pixel in each of the N output planes, a plane-specific scalar bias is added, and a nonlinearity is applied. The resulting planes are known as *feature maps*. Convolutional neural networks repeat this structure, often interleaved with a spatial decimation operation such as taking the average or maximum value in a given neighbourhood (*average pooling* or *max pooling*, respectively), though more recently simple subsampling has become a popular alternative (Springenberg et al., 2015). If a discrete convolution is immediately followed by a fixed subsampling, the composed operation can be computationally streamlined by never computing the outputs of the convolution which will be immediately discarded. The composed operation is often referred to as *strided convolution*.[ii]

Convolutional layers involve some subtleties with respect to the treatment of image borders, particularly when "transposing" the convolution for the computation of gradients. We refer the reader to Dumoulin and Visin (2016) for a thorough treatment of the subject as it applies to neural networks in practice.

1.3.3 Unsupervised learning

We now turn our attention to the problem of performing unsupervised learning using neural networks.

Most modern methods for unsupervised deep learning have a straightforward

i. Equivalent up to a horizontal and vertical reversal of the weights, which is inconsequential if these weights are being fit to data rather than given *a priori*.

ii. N.B.: While the gradient of a convolution is always expressible as another convolution with a different treatment of the image border, the gradient of this combined operation is *not* expressible as a convolution, but rather a spatial dilation via the insertion of zeros, followed by a convolution. This "strided convolution-transpose", "fractionally strided convolution", or "up-convolution" operation is often used in image generation architectures, such as those employed in chapter 10.

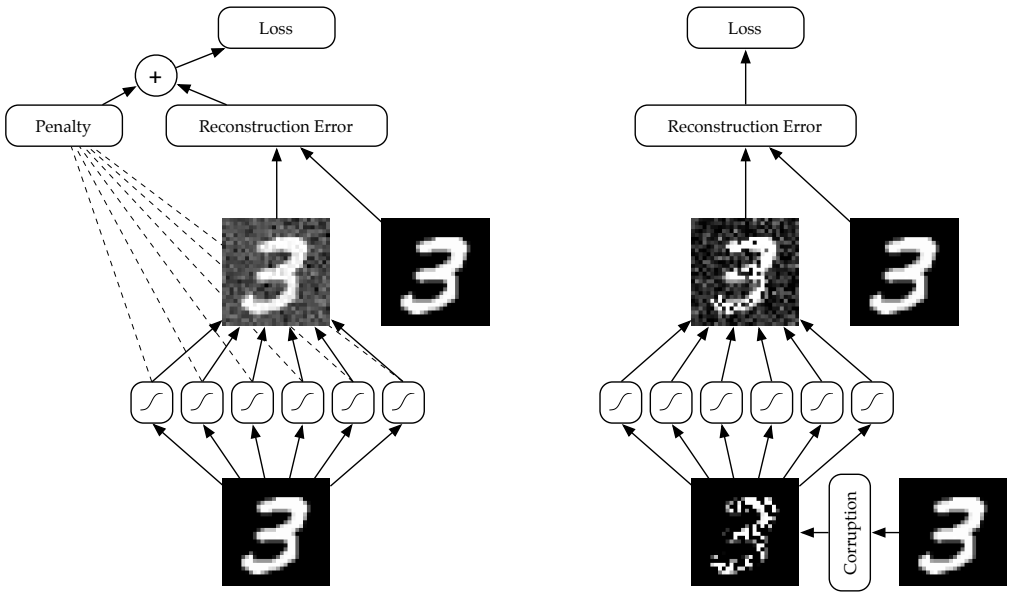

Figure 1.5 – A schematic diagram of a penalized autoencoder (left) and a denoising autoencoder (right).

interpretation in terms of probabilistic graphical models. While much of the early work on deep unsupervised learning relied heavily upon the machinery of *undirected* graphical models (Hinton et al., 2006; Hinton and Salakhutdinov, 2006; Salakhutdinov and Hinton, 2009), more recent work bridging deep neural networks and probabilistic models has focused on directed latent variable models (Gregor et al., 2014; Mnih and Gregor, 2014; Rezende et al., 2014; Kingma and Welling, 2014; Goodfellow et al., 2014; Dinh et al., 2016) and fully-observed, auto-regressive models (Larochelle and Murray, 2011; Germain et al., 2015; Oord et al., 2016).

In the interest of conciseness, we review only the concepts necessary to elucidate the contributions of this thesis. We review the venerable *autoencoder*, a deterministic model which nonetheless underpins many modern probabilistically oriented techniques, and the *denoising autoencoder*, which admits an unconventional probabilistic interpretation. We further discuss generative adversarial networks (Goodfellow et al., 2014) in chapter 8.

Autoencoders

An *autoencoder* is a deterministic feed-forward neural network, i.e. akin to a multilayer perceptron, that is trained to reproduce its input in the output layer (rather than predict some target or response value). A simple single-layer autoencoder might be parameterized as an encoder function h and a decoder function g

$$h(\mathbf{x}) = s(\mathbf{W}\mathbf{x} + \mathbf{b}) \tag{1.13}$$

$$g(\mathbf{x}) = t(\mathbf{V}\mathbf{x} + \mathbf{c}) \tag{1.14}$$

where s and t are elementwise activation functions and trained such that $g(h(\mathbf{x})) \approxeq \mathbf{x}$ using a loss function appropriate for the domain of the inputs (and activation functions). In the case real-valued inputs with an assumption of independent Gaussian noise, squared Euclidean distance ("mean squared error") between the input and the output vector is a reasonable choice; in the case of pseudo-binary inputs in $[0, 1]^D$ and $t(\cdot) = \sigma(\cdot)$, a reasonable choice may be the cross-entropy (discussed in 1.1) when treating the inputs and outputs as the parameters of independent Bernoulli distributions. Autoencoders are closely related to Principal Components Analysis, an unsupervised dimensionality reduction method (Jolliffe, 1986): in particular, an autoencoder is equivalent to PCA when trained with mean squared error, a number of hidden units K less than the number of input dimensions D, and s and t equal to the identity function (Baldi and Hornik, 1989). The columns of $\mathbf{W}$ will span the same subspace as the first K principal components, but will not form an orthonormal set.

In early work, authors focused on the case of fewer hidden units than input variables (Bourlard and Kamp, 1988), and a single hidden layer – autoencoders with many layers of nonlinear hidden units were traditionally considered difficult to train, though layerwise pre-training with RBMs (Hinton and Salakhutdinov, 2006) and more sophisticated optimization strategies (Martens, 2010) have yielded successes in training deep autoencoders with a "bottleneck".

In modern approaches, the *over-complete* (more hidden units than input dimensions) setting is often preferred, but additional constraints or penalties are necessary in order to prevent trivial solutions: with more hidden units than feature

dimensions, there may be arbitrarily many mappings that reconstruct the input perfectly but fail to capture any interesting structure.

Denoising autoencoders

Vincent et al. (2008) proposed an alternate method of constraining the representation: corrupt each training example according to some noise process, and train the *denoising autoencoder* to reconstruct the original example from its corrupted counterpart. The same work showed that stacking these modules yields performance competitive with deep belief networks (Hinton et al., 2006), while Vincent (2011) showed a theoretical connection between single hidden layer denoising autoencoders (trained with a squared error loss) and restricted Boltzmann machines trained with an alternative to maximum likelihood called score matching (Hyvärinen, 2005). Bengio et al. (2013) expanded upon this connection, proposing an interpretation of denoising autoencoders as generative models. This interpretation applies to arbitrary architectures and any loss function interpretable as a negative log likelihood.

2 Prologue to First Article

2.1 Article Details

Maxout networks. Ian J. Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron Courville, and Yoshua Bengio. *Proceedings of the 30th International Conference on Machine Learning (ICML '13)*, pp. 1319-1327.

Personal Contribution. Ian Goodfellow and I jointly undertook engineering work (wrapping of the `cuda-convnet` library for use with Theano) for the implementation of the large scale experiments. I proposed and implemented many of the probative experiments in sections 6 through 8, and performed the majority of the benchmark experiments on CIFAR10/CIFAR100. I co-wrote the manuscript with the other authors.

2.2 Context

A turning point in the adoption of deep learning methods came in 2012, when a convolutional neural network won the ImageNet Large Scale Visual Recognition Challenge (Krizhevsky et al., 2012). Two important components in the design of this network were the use of rectified linear activations (Jarrett et al., 2009; Nair and Hinton, 2010; Glorot et al., 2011) and the dropout method (Hinton et al., 2012) for regularization. Observations that dropout regularization appeared to be most effective when used in conjunction with rectified linear activations led to the question of whether other piece-wise affine parameterizations would offer yet greater synergy with dropout.

2.3 Contributions

Maxout units offer a novel alternative to traditional elementwise activations, removing the saturating property of rectified linear units. We show that networks of maxout units improve upon several important classification benchmarks, and conduct extensive experiments to explain the improved performance.

2.4 Recent Developments

As of this writing, the manuscript has accrued over 800 citations. Maxout units have been successfully leveraged in a variety of applications of neural networks, including automatic speech recognition (Swietojanski et al., 2014; Zhang et al., 2014), automated speaker verification (Variani et al., 2014), whale call detection (Smirnov, 2013), face recognition (Schroff et al., 2015), visual person reidentification (Li et al., 2014), house number transcription from photographs (Goodfellow et al., 2014), and brain tumour segmentation (Havaei et al., 2017).

3 Maxout Networks

3.1 Introduction

Dropout (Hinton et al., 2012) provides an inexpensive and simple means of both training a large ensemble of models that share parameters and approximately averaging together these models' predictions. Dropout applied to multilayer perceptrons and deep convolutional networks has improved the state of the art on tasks ranging from audio classification to very large scale object recognition (Hinton et al., 2012; Krizhevsky et al., 2012). While dropout is known to work well in practice, it has not previously been demonstrated to actually perform model averaging for deep architectures [i] . Dropout is generally viewed as an indiscriminately applicable tool that reliably yields a modest improvement in performance when applied to almost any model.

We argue that rather than using dropout as a slight performance enhancement applied to arbitrary models, the best performance may be obtained by directly designing a model that enhances dropout's abilities as a model averaging technique. Training using dropout differs significantly from previous approaches such as ordinary stochastic gradient descent. Dropout is most effective when taking relatively large steps in parameter space. In this regime, each update can be seen as making a significant update to a different model on a different subset of the training set. The ideal operating regime for dropout is when the overall training procedure resembles training an ensemble with bagging under parameter sharing constraints. This differs radically from the ideal stochastic gradient operating regime in which a single model makes steady progress via small steps. Another consideration is that dropout model averaging is only an approximation when applied to deep models. Explicitly designing models to minimize this approximation error may thus enhance dropout's performance as well.

i. Between submission and publication of this paper, we have learned that Srivastava (2013) performed experiments on this subject similar to ours.

We propose a simple model that we call *maxout* that has beneficial characteristics both for optimization and model averaging with dropout. We use this model in conjunction with dropout to set the state of the art on four benchmark datasets [i].

3.2 Review of dropout

Dropout is a technique that can be applied to deterministic feed-forward architectures that predict an output y given input vector v. These architectures contain a series of hidden layers $\mathbf{h} = \{h^{(1)}, \ldots, h^{(L)}\}$. Dropout trains an ensemble of models consisting of the set of all models that contain a subset of the variables in both v and $\mathbf{h}$. The same set of parameters θ is used to parameterize a family of distributions $p(y \mid v; \theta, \mu)$ where $\mu \in \mathcal{M}$ is a binary mask determining which variables to include in the model. On each presentation of a training example, we train a different sub-model by following the gradient of $\log p(y \mid v; \theta, \mu)$ for a different randomly sampled μ. For many parameterizations of p (such as most multilayer perceptrons) the instantiation of different sub-models $p(y \mid v; \theta, \mu)$ can be obtained by elementwise multiplication of v and $\mathbf{h}$ with the mask μ. Dropout training is similar to bagging (Breiman, 1994), where many different models are trained on different subsets of the data. Dropout training differs from bagging in that each model is trained for only one step and all of the models share parameters. For this training procedure to behave as if it is training an ensemble rather than a single model, each update must have a large effect, so that it makes the sub-model induced by that μ fit the current input v well.

The functional form becomes important when it comes time for the ensemble to make a prediction by averaging together all the sub-models' predictions. Most prior work on bagging averages with the arithmetic mean, but it is not obvious how to do so with the exponentially many models trained by dropout. Fortunately, some model families yield an inexpensive *geometric* mean. When $p(y \mid v; \theta) = \text{softmax}(v^\top W + b)$, the predictive distribution defined by renormalizing the geometric mean of $p(y \mid v; \theta, \mu)$ over $\mathcal{M}$ is simply given by $\text{softmax}(v^\top W/2 + b)$.

i. Code and hyperparameters available at http://www-etud.iro.umontreal.ca/~goodfeli/maxout.html

In other words, the average prediction of exponentially many sub-models can be computed simply by running the full model with the weights divided by 2. This result holds exactly in the case of a single layer softmax model. Previous work on dropout applies the same scheme in deeper architectures, such as multilayer perceptrons, where the $W/2$ method is only an approximation to the geometric mean. The approximation has not been characterized mathematically, but performs well in practice.

3.3 Description of maxout

The maxout model is simply a feed-forward achitecture, such as a multilayer perceptron or deep convolutional neural network, that uses a new type of activation function: the maxout unit. Given an input $x \in \mathbb{R}^d$ (x may be v, or may be a hidden layer's state), a maxout hidden layer implements the function

$$h_i(x) = \max_{j \in [1,k]} z_{ij}$$

where $z_{ij} = x^\top W_{\ldots ij} + b_{ij}$, and $W \in \mathbb{R}^{d \times m \times k}$ and $b \in \mathbb{R}^{m \times k}$ are learned parameters. In a convolutional network, a maxout feature map can be constructed by taking the maximum across k affine feature maps (i.e., pool across channels, in addition spatial locations). When training with dropout, we perform the elementwise multiplication with the dropout mask immediately prior to the multiplication by the weights in all cases–we do not drop inputs to the max operator. A single maxout unit can be interpreted as making a piecewise linear approximation to an arbitrary convex function. Maxout networks learn not just the relationship between hidden units, but also the activation function of each hidden unit. See Fig. 3.1 for a graphical depiction of how this works.

Maxout abandons many of the mainstays of traditional activation function design. The representation it produces is not sparse at all (see Fig. 3.2), though the gradient is highly sparse and dropout will artificially sparsify the effective representation during training. While maxout may learn to saturate on one side or the other this is a measure zero event (so it is almost never bounded from above). While a significant proportion of parameter space corresponds to the function being

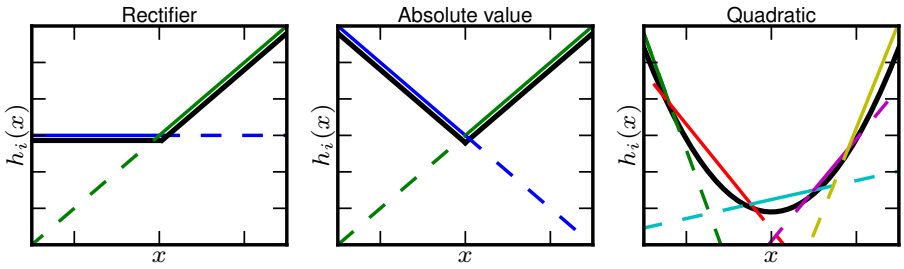

Figure 3.1 – Graphical depiction of how the maxout activation function can implement the rectified linear, absolute value rectifier, and approximate the quadratic activation function. This diagram is 2D and only shows how maxout behaves with a 1D input, but in multiple dimensions a maxout unit can approximate arbitrary convex functions.

bounded from below, maxout is not constrained to learn to be bounded at all. Maxout is locally linear almost everywhere, while many popular activation functions have signficant curvature. Given all of these departures from standard practice, it may seem surprising that maxout activation functions work at all, but we find that they are very robust and easy to train with dropout, and achieve excellent performance.

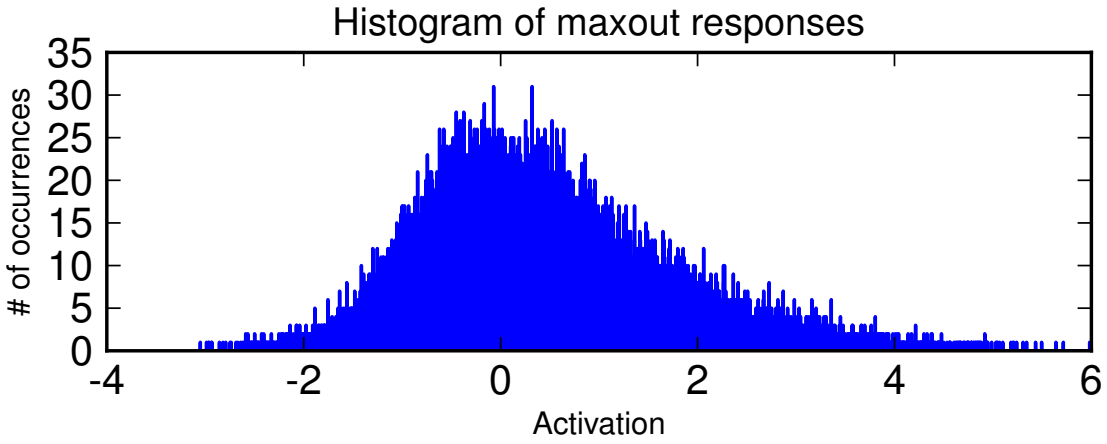

Figure 3.2 – The activations of maxout units are not sparse.

3.4 Maxout is a universal approximator

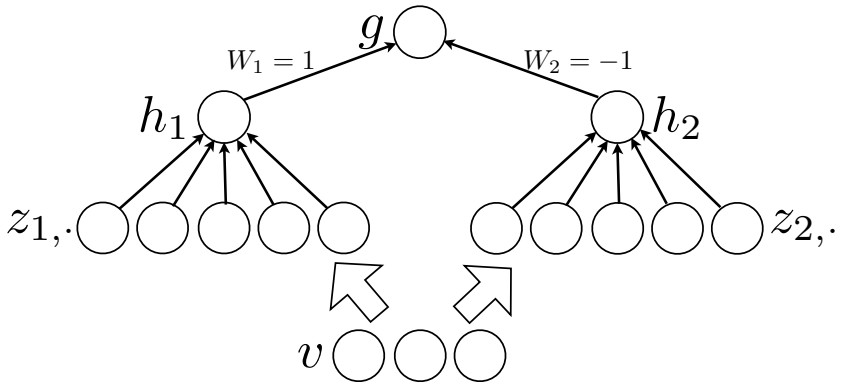

Figure 3.3 – An MLP containing two maxout units can arbitrarily approximate any continuous function. The weights in the final layer can set g to be the difference of h_1 and h_2. If z_1 and z_2 are allowed to have arbitrarily high cardinality, h_1 and h_2 can approximate any convex function. g can thus approximate any continuous function due to being a difference of approximations of arbitrary convex functions.

A standard MLP with enough hidden units is a universal approximator. Similarly, maxout networks are universal approximators. Provided that each individual maxout unit may have arbitrarily many affine components, we show that a maxout model with just two hidden units can approximate, arbitrarily well, any continuous function of $v \in \mathbb{R}^n$. A diagram illustrating the basic idea of the proof is presented in Fig. 3.3.

Consider the continuous piecewise linear (PWL) function $g(v)$ consisting of k locally affine regions on $\mathbb{R}^n$.

Proposition 3.4.1. *(From Theorem 2.1 in Wang (2004)) For any positive integers m and n, there exist two groups of $n+1$-dimensional real-valued parameter vectors $[W_{1j}, b_{1j}], j \in [1, k]$ and $[W_{2j}, b_{2j}], j \in [1, k]$ such that:*

$$g(v) = h_1(v) - h_2(v) \tag{3.1}$$

That is, any continuous PWL function can be expressed as a difference of two convex PWL functions. The proof is given in Wang (2004).

Proposition 3.4.2. *From the* Stone-Weierstrass approximation theorem*, let C be a compact domain $C \subset \mathbb{R}^n$, $f : C \to \mathbb{R}$ be a continuous function, and $\epsilon > 0$ be any*

positive real number. Then there exists a continuous PWL function g, (depending upon ϵ), such that for all $v \in C$, $|f(v) - g(v)| < \epsilon$.

Theorem 3.4.3. Universal approximator theorem. *Any continuous function f can be approximated arbitrarily well on a compact domain $C \subset \mathbb{R}^n$ by a maxout network with two maxout hidden units.*

Sketch of Proof By Proposition 3.4.2, any continuous function can be approximated arbitrarily well (up to ϵ), by a piecewise linear function. We now note that the representation of piecewise linear functions given in Proposition 3.4.1 exactly matches a maxout network with two hidden units $h_1(v)$ and $h_2(v)$, with sufficiently large k to achieve the desired degree of approximation ϵ. Combining these, we conclude that a two hidden unit maxout network can approximate any continuous function $f(v)$ arbitrarily well on the compact domain C. In general as $\epsilon \to 0$, we have $k \to \infty$.

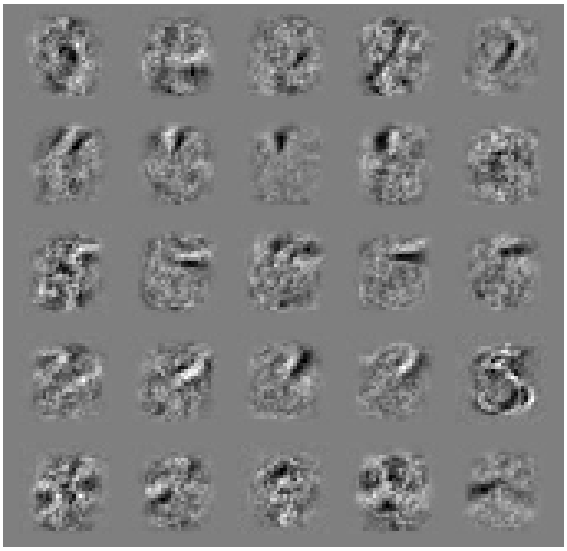

Figure 3.4 – Example filters learned by a maxout MLP trained with dropout on MNIST. Each row contains the filters whose responses are pooled to form a maxout unit.

Table 3.1 – Test set misclassification rates for the best methods on the permutation invariant MNIST dataset. Only methods that are regularized by modeling the input distribution outperform the maxout MLP.

Method	Test error
Rectifier MLP + dropout (Srivastava, 2013)	1.05%
DBM (Salakhutdinov and Hinton, 2009)	0.95%
Maxout MLP + dropout	**0.94%**
MP-DBM (Goodfellow et al., 2013)	0.88%
Deep Convex Network (Yu and Deng, 2011)	0.83%
Manifold Tangent Classifier (Rifai et al., 2011)	0.81%
DBM + dropout (Hinton et al., 2012)	0.79%

3.5 Benchmark results

We evaluated the maxout model on four benchmark datasets and set the state of the art on all of them.

3.5.1 MNIST

The MNIST (LeCun et al., 1998) dataset consists of 28 × 28 pixel greyscale images of handwritten digits 0-9, with 60,000 training and 10,000 test examples. For the *permutation invariant* version of the MNIST task, only methods unaware of the 2D structure of the data are permitted. For this task, we trained a model consisting of two densely connected maxout layers followed by a softmax layer. We regularized the model with dropout and by imposing a constraint on the norm of each weight vector, as in (Srebro and Shraibman, 2005). Apart from the maxout units, this is the same architecture used by Hinton et al. (2012). We selected the hyperparameters by minimizing the error on a validation set consisting of the last 10,000 training examples. To make use of the full training set, we recorded the

Table 3.2 – Test set misclassification rates for the best methods on the general MNIST dataset, excluding methods that augment the training data.

Method	Test error
2-layer CNN+2-layer NN (Jarrett et al., 2009)	0.53%
Stochastic pooling Zeiler and Fergus (2013)	0.47%
Conv. maxout + dropout	**0.45%**

value of the log likelihood on the first 50,000 examples at the point of minimal validation error. We then continued training on the full 60,000 example training set until the validation set log likelihood matched this number. We obtained a test set error of 0.94%, which is the best result we are aware of that does not use unsupervised pretraining. We summarize the best published results on permutation invariant MNIST in Table 3.1.

We also considered the MNIST dataset without the permutation invariance restriction. In this case, we used three convolutional maxout hidden layers (with spatial max pooling on top of the maxout layers) followed by a densely connected softmax layer. We were able to rapidly explore hyperparameter space thanks to the extremely fast GPU convolution library developed by Krizhevsky et al. (2012). We obtained a test set error rate of 0.45%, which sets a new state of the art in this category. (It is possible to get better results on MNIST by augmenting the dataset with transformations of the standard set of images (Ciresan et al., 2010).) A summary of the best methods on the general MNIST dataset is provided in Table 3.2.

3.5.2 CIFAR-10

The CIFAR-10 dataset (Krizhevsky and Hinton, 2009) consists of 32×32 color images drawn from 10 classes split into 50,000 train and 10,000 test images. We preprocess the data using global contrast normalization and ZCA whitening.

We follow a similar procedure as with the MNIST dataset, with one change. On MNIST, we find the best number of training epochs in terms of validation set error, then record the training set log likelihood and continue training using the entire training set until the validation set log likelihood has reached this value. On

Table 3.3 – Test set misclassification rates for the best methods on the CIFAR-10 dataset.

Method	Test error
Stochastic pooling Zeiler and Fergus (2013)	15.13%
CNN + Spearmint Snoek et al. (2012)	14.98%
Conv. maxout + dropout	**11.68 %**
CNN + Spearmint + data augmentation Snoek et al. (2012)	9.50 %
Conv. maxout + dropout + data augmentation	**9.38 %**

CIFAR-10, continuing training in this fashion is infeasible because the final value of the learning rate is very small and the validation set error is very high. Training until the validation set likelihood matches the cross-validated value of the training likelihood would thus take prohibitively long. Instead, we retrain the model from scratch, and stop when the new likelihood matches the old one.

Our best model consists of three convolutional maxout layers, a fully connected maxout layer, and a fully connected softmax layer. Using this approach we obtain a test set error of 11.68%, which improves upon the state of the art by over two percentage points. (If we do not train on the validation set, we obtain a test set error of 13.2%, which also improves over the previous state of the art.) If we additionally augment the data with translations and horizontal reflections, we obtain the absolute state of the art on this task at 9.35% error. In this case, the likelihood during the retrain never reaches the likelihood from the validation run, so we retrain for the same number of epochs as the validation run. A summary of the best CIFAR-10 methods is provided in Table 3.3.

3.5.3 CIFAR-100

The CIFAR-100 (Krizhevsky and Hinton, 2009) dataset is the same size and format as the CIFAR-10 dataset, but contains 100 classes, with only one tenth as many labeled examples per class. Due to lack of time we did not extensively cross-

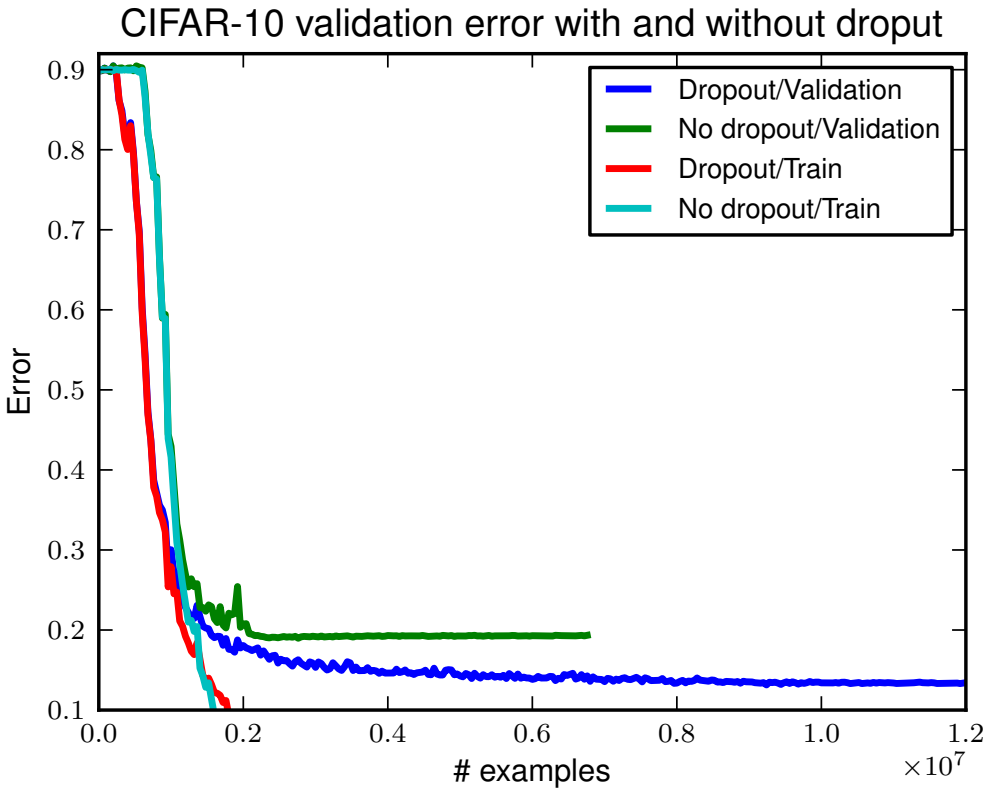

Figure 3.5 – When training maxout, the improvement in validation set error that results from using dropout is dramatic. Here we find a greater than 25% reduction in our validation set error on CIFAR-10.

validate hyperparameters on CIFAR-100 but simply applied hyperparameters we found to work well on CIFAR-10. We obtained a test set error of 38.57%, which is state of the art. If we do not retrain using the entire training set, we obtain a test set error of 41.48%, which also surpasses the current state of the art. A summary of the best methods on CIFAR-100 is provided in Table 3.4.

3.5.4 Street View House Numbers

The SVHN (Netzer et al., 2011) dataset consists of color images of house numbers collected by Google Street View. The dataset comes in two formats. We consider the second format, in which each image is of size 32×32 and the task is to classify the digit in the center of the image. Additional digits may appear beside it but must be ignored. There are 73,257 digits in the training set, 26,032 digits in the test set and 531,131 additional, somewhat less difficult examples, to use as an

Table 3.4 – Test set misclassification rates for the best methods on the CIFAR-100 dataset.

Method	Test error
Learned pooling (Malinowski and Fritz, 2013)	43.71%
Stochastic poolingZeiler and Fergus (2013)	42.51%
Conv. maxout + dropout	**38.57%**

Table 3.5 – Test set misclassification rates for the best methods on the SVHN dataset.

Method	Test error
Sermanet et al. (2012)	4.90%
Stochastic pooling Zeiler and Fergus (2013)	2.80 %
Rectifiers + dropout Srivastava (2013)	2.78 %
Rectifiers + dropout + synthetic translation Srivastava (2013)	2.68 %
Conv. maxout + dropout	**2.47 %**

extra training set. Following Sermanet et al. (2012), to build a validation set, we select 400 samples per class from the training set and 200 samples per class from the extra set. The remaining digits of the train and extra sets are used for training.

For SVHN, we did not train on the validation set at all. We used it only to find the best hyperparameters. We applied local contrast normalization preprocessing the same way as Zeiler and Fergus (2013). Otherwise, we followed the same approach as on MNIST. Our best model consists of three convolutional maxout hidden layers and a densely connected maxout layer followed by a densely connected softmax layer. We obtained a test set error rate of 2.47%, which sets the state of the art. A summary of comparable methods is provided in Table 3.5.

3.6 Comparison to rectifiers

One obvious question about our results is whether we obtained them by improved preprocessing or larger models, rather than by the use of maxout. For MNIST we used no preprocessing, and for SVHN, we use the same preprocessing as Zeiler and Fergus (2013). However on the CIFAR datasets we did use a new form of preprocessing. We therefore compare maxout to rectifiers run with the same processing and a variety of model sizes on this dataset.

By running a large cross-validation experiment (see Fig. 3.6) we found that maxout offers a clear improvement over rectifiers. We also found that our preprocessing and size of models improves rectifiers and dropout beyond the previous state of the art result. Cross-channel pooling is a method for reducing the size of state and number of parameters needed to have a given number of filters in the model. Performance seems to correlate well with the number of filters for maxout but with the number of output units for rectifiers–i.e, rectifier units do not benefit much from cross-channel pooling. Rectifier units do best without cross-channel pooling but with the same number of filters, meaning that the size of the state and the number of parameters must be about k times higher for rectifiers to obtain generalization performance approaching that of maxout.

3.7 Model averaging

Having demonstrated that maxout networks are effective models, we now analyze the reasons for their success. We first identify reasons that maxout is highly compatible with dropout's approximate model averaging technique.

The intuitive justification for averaging sub-models by dividing the weights by 2 given by (Hinton et al., 2012) is that this does exact model averaging for a single layer model, softmax regression. To this characterization, we add the observation that the model averaging remains exact if the model is extended to multiple linear layers. While this has the same representational power as a single layer, the expression of the weights as a product of several matrices could have a different inductive bias. More importantly, it indicates that dropout does exact model averaging in deeper architectures provided that they are locally linear among

the space of inputs to each layer that are visited by applying different dropout masks.

We argue that dropout training encourages maxout units to have large linear regions around inputs that appear in the training data. Because each sub-model must make a good prediction of the output, each unit should learn to have roughly the same activation regardless of which inputs are dropped. In a maxout network with arbitrarily selected parameters, varying the dropout mask will often move the effective inputs far enough to escape the local region surrounding the clean inputs in which the hidden units are linear, i.e., changing the dropout mask could frequently change which piece of the piecewise function an input is mapped to. Maxout *trained with dropout* may have the identity of the maximal filter in each unit change relatively rarely as the dropout mask changes. Networks of linear operations and $\max(\cdot)$ may learn to exploit dropout's approximate model averaging technique well.

Many popular activation functions have significant curvature nearly everywhere. These observations suggest that the approximate model averaging of dropout will not be as accurate for networks incorporating such activation functions. To test this, we compared the best maxout model trained on MNIST with dropout to a hyperbolic tangent network trained on MNIST with dropout. We sampled several subsets of each model and compared the geometric mean of these sampled models' predictions to the prediction made using the dropout technique of dividing the weights by 2. We found evidence that dropout is indeed performing model averaging, even in multilayer networks, and that it is more accurate in the case of maxout. See Fig. 3.7 and Fig. 3.8 for details.

3.8 Optimization

The second key reason that maxout performs well is that it improves the bagging style training phase of dropout. Note that the arguments in section 3.7 motivating the use of maxout also apply equally to rectified linear units (Salinas and Abbott, 1996; Hahnloser, 1998; Glorot et al., 2011). The only difference between maxout and max pooling over a set of rectified linear units is that maxout does not include a 0 in the max. Superficially, this seems to be a small difference, but we find that

including this constant 0 is very harmful to optimization in the context of dropout. For instance, on MNIST our best validation set error with an MLP is 1.04%. If we include a 0 in the max, this rises to over 1.2%. We argue that, when trained with dropout, maxout is easier to optimize than rectified linear units with cross-channel pooling.

3.8.1 Optimization experiments

To verify that maxout yields better optimization performance than max pooled rectified linear units when training with dropout, we carried out two experiments. First, we stressed the optimization capabilities of the training algorithm by training a small (two hidden convolutional layers with $k = 2$ and sixteen kernels) model on the large (600,000 example) SVHN dataset. When training with rectifier units the training error gets stuck at 7.3%. If we train instead with maxout units, we obtain 5.1% training error. As another optimization stress test, we tried training very deep and narrow models on MNIST, and found that maxout copes better with increasing depth than pooled rectifiers. See Fig. 3.9 for details.

3.8.2 Saturation

Optimization proceeds very differently when using dropout than when using ordinary stochastic gradient descent. SGD usually works best with a small learning rate that results in a smoothly decreasing objective function, while dropout works best with a large learning rate, resulting in a constantly fluctuating objective function. Dropout rapidly explores many different directions and rejects the ones that worsen performance, while SGD moves slowly and steadily in the most promising direction. We find empirically that these different operating regimes result in different outcomes for rectifier units. When training with SGD, we find that the rectifier units saturate at 0 less than 5% of the time. When training with dropout, we initialize the units to sature rarely but training gradually increases their saturation rate to 60%. Because the 0 in the $\max(0, z)$ activation function is a constant, this blocks the gradient from flowing through the unit. In the absence of gradient through the unit, it is difficult for training to change this unit to become active again. *Maxout does not suffer from this problem because gradient always flows through every maxout unit*–even when a maxout unit is 0, this 0 is a function

of the parameters and may be adjusted. Units that take on negative activations may be steered to become positive again later. Fig. 3.10 illustrates how active rectifier units become inactive at a greater rate than inactive units become active when training with dropout, but maxout units, which are always active, transition between positive and negative activations at about equal rates in each direction. We hypothesize that the high proportion of zeros and the difficulty of escaping them impairs the optimization performance of rectifiers relative to maxout.

To test this hypothesis, we trained two MLPs on MNIST, both with two hidden layers and 1200 filters per layer pooled in groups of 5. When we include a constant 0 in the max pooling, the resulting trained model fails to make use of 17.6% of the filters in the second layer and 39.2% of the filters in the second layer. A small minority of the filters usually took on the maximal value in the pool, and the rest of the time the maximal value was a constant 0. Maxout, on the other hand, used all but 2 of the 2400 filters in the network. Each filter in each maxout unit in the network was maximal for some training example. All filters had been utilised and tuned.

3.8.3 Lower layer gradients and bagging

To behave differently from SGD, dropout requires the gradient to change noticeably as the choice of which units to drop changes. If the gradient is approximately constant with respect to the dropout mask, then dropout simplifies to SGD training. We tested the hypothesis that rectifier networks suffer from diminished gradient flow to the lower layers of the network by monitoring the variance with respect to dropout masks for fixed data during training of two different MLPs on MNIST. The variance of the gradient on the output weights was 1.4 times larger for maxout on an average training step, while the variance on the gradient of the first layer weights was 3.4 times larger for maxout than for rectifiers. Combined with our previous result showing that maxout allows training deeper networks, this greater variance suggests that maxout better propagates varying information downward to the lower layers and helps dropout training to better resemble bagging for the lower-layer parameters. Rectifier networks, with more of their gradient lost to saturation, presumably cause dropout training to resemble regular SGD toward the bottom of the network.

3.9 Conclusion

We have proposed a new activation function called maxout that is particularly well suited for training with dropout, and for which we have proven a universal approximation theorem. We have shown empirical evidence that dropout attains a good approximation to model averaging in deep models. We have shown that maxout exploits this model averaging behavior because the approximation is more accurate for maxout units than for tanh units. We have demonstrated that optimization behaves very differently in the context of dropout than in the pure SGD case. By designing the maxout gradient to avoid pitfalls such as failing to use many of a model's filters, we are able to train deeper networks than is possible using rectifier units. We have also shown that maxout propagates variations in the gradient due to different choices of dropout masks to the lowest layers of a network, ensuring that every parameter in the model can enjoy the full benefit of dropout and more faithfully emulate bagging training. The state of the art performance of our approach on five different benchmark tasks motivates the design of further models that are explicitly intended to perform well when combined with inexpensive approximations to model averaging.

Acknowledgements

The authors would like to thank the developers of Theano (Bergstra et al., 2010; Bastien et al., 2012), in particular Frédéric Bastien and Pascal Lamblin for their assistance with infrastructure development and performance optimization. We would also like to thank Yann Dauphin for helpful discussions.

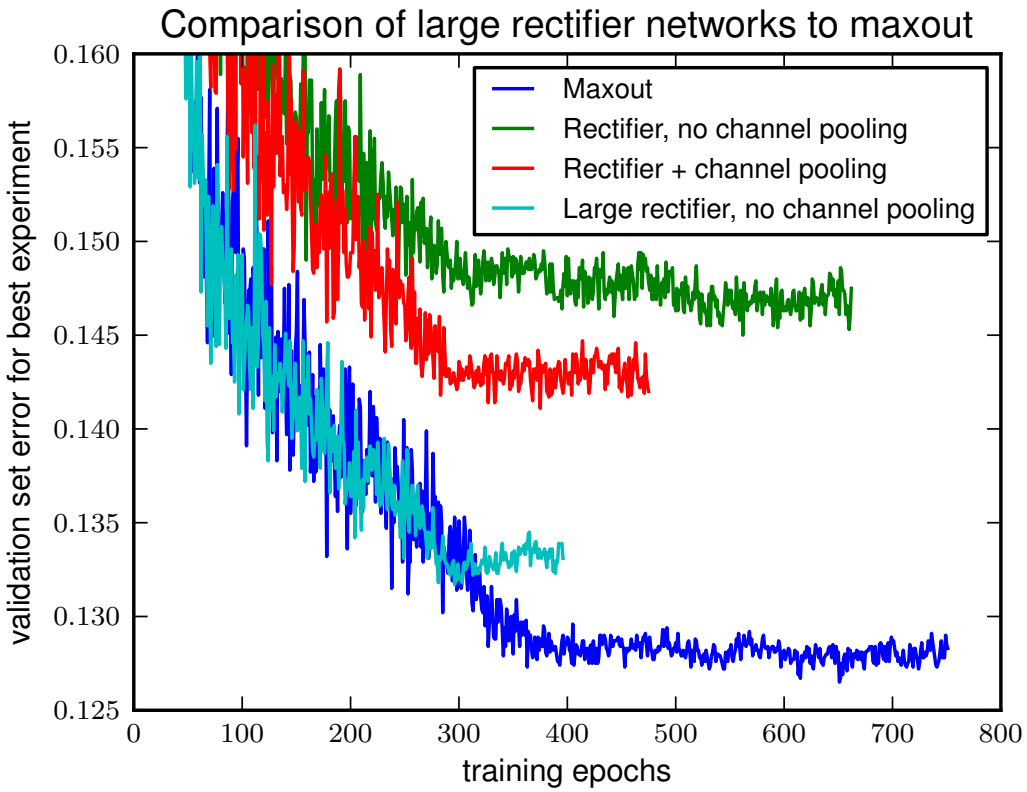

Figure 3.6 – We cross-validated the momentum and learning rate for four architectures of model: 1) Medium-sized maxout network. 2) Rectifier network with cross-channel pooling, and exactly the same number of parameters and units as the maxout network. 3) Rectifier network without cross-channel pooling, and the same number of units as the maxout network (thus fewer parameters). 4) Rectifier network without cross-channel pooling, but with k times as many units as the maxout network. Because making layer i have k times more outputs increases the number of inputs to layer $i + 1$, this network has roughly k times more parameters than the maxout network, and requires significantly more memory and runtime. We sampled 10 learning rate and momentum schedules and random seeds for dropout, then ran each configuration for all 4 architectures. Each curve terminates after failing to improve the validation error in the last 100 epochs.

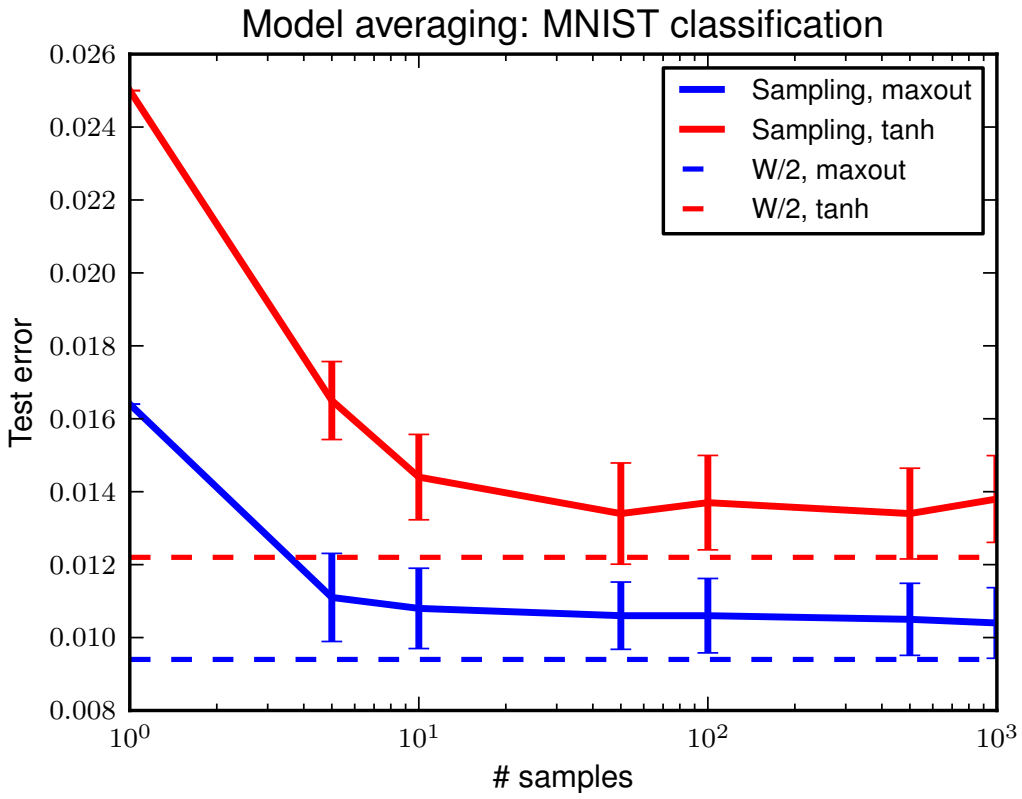

Figure 3.7 – The error rate of the prediction obtained by sampling several sub-models and taking the geometric mean of their predictions approaches the error rate of the prediction made by dividing the weights by 2. However, the divided weights still obtain the best test error, suggesting that dropout is a good approximation to averaging over a very large number of models. Note that the correspondence is more clear in the case of maxout.

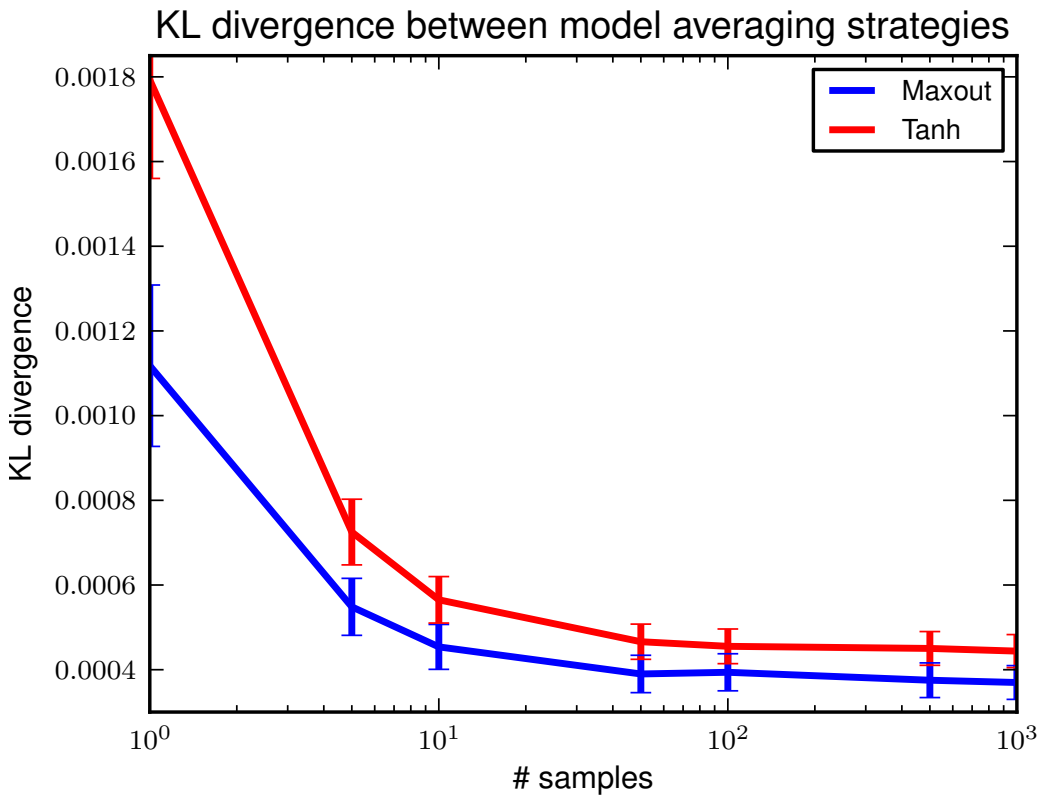

Figure 3.8 – The KL divergence between the distribution predicted using the dropout technique of dividing the weights by 2 and the distribution obtained by taking the geometric mean of the predictions of several sampled models decreases as the number of samples increases. This suggests that dropout does indeed do model averaging, even for deep networks. The approximation is more accurate for maxout units than for tanh units.

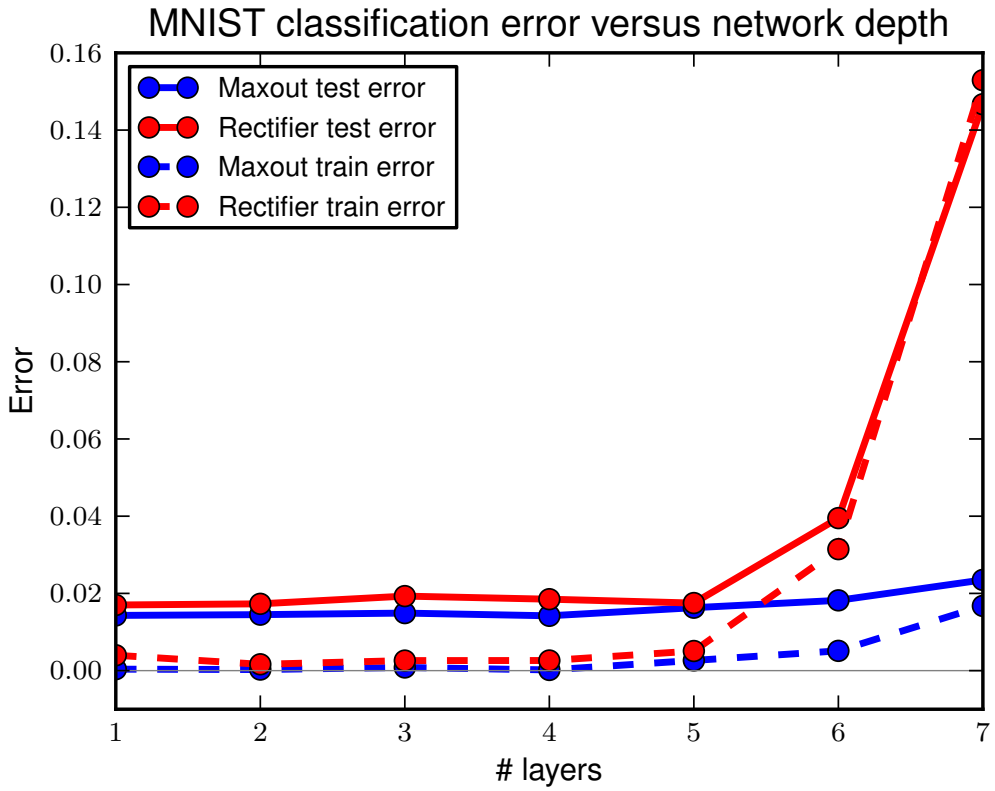

Figure 3.9 – We trained a series of models with increasing depth on MNIST. Each layer contains only 80 units (k=5) to make fitting the training set difficult. Maxout optimization degrades gracefully with depth but pooled rectifier units worsen noticeably at 6 layers and dramatically at 7.

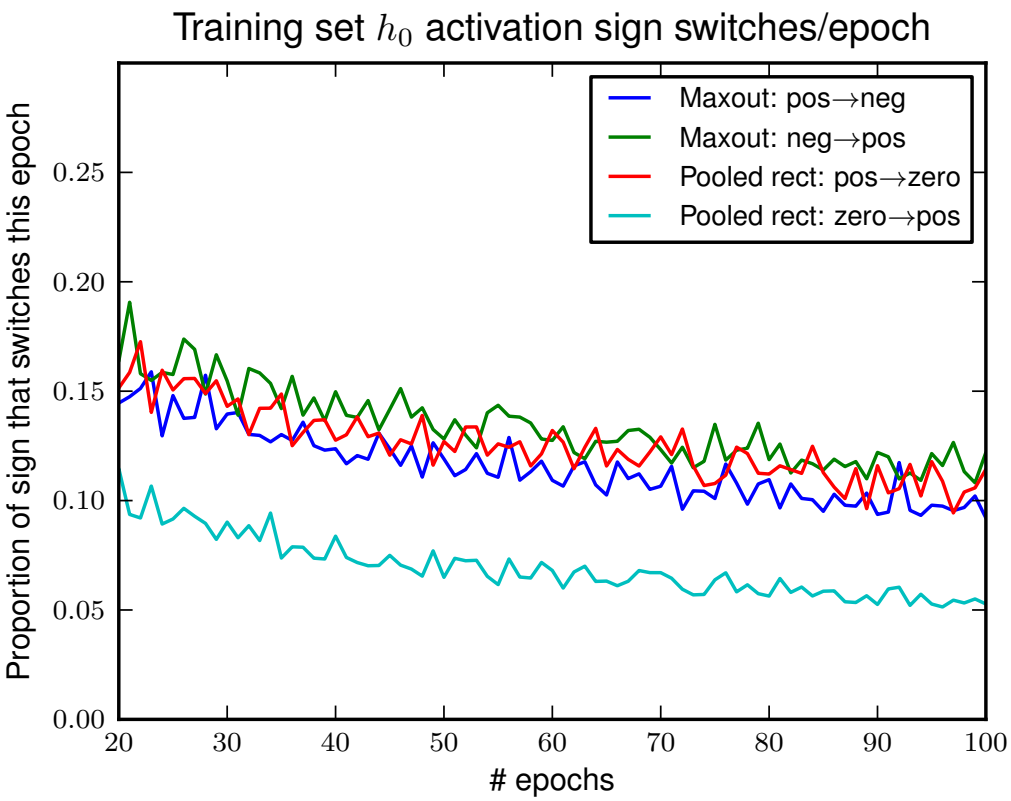

Figure 3.10 – During dropout training, rectifier units transition from positive to 0 activation more frequently than they make the opposite transition, resulting a preponderence of 0 activations. Maxout units freely move between positive and negative signs at roughly equal rates.

4 Prologue to Second Article

4.1 Article Details

An empirical analysis of dropout in piecewise linear networks. David Warde-Farley, Ian Goodfellow, Aaron Courville, and Yoshua Bengio. *Proceedings of the Second International Conference on Learning Representatios (ICLR '14).*

Personal Contribution.

I conceived of all of the experiments in this chapter, either alone or in collaboration with my co-authors. I devised the probe tasks, and ran all experiments. Ian Goodfellow and I wrote the manuscript.

4.2 Context

After its introduction, dropout (Hinton et al., 2012) was becoming a popular method for regularizing neural networks but was very poorly understood. Many theories competed to explain its remarkable efficacy, and questions remained about the weight-scaling heuristic used at test time. The authors put forth an explanation in terms of an exponentially large ensemble of networks which share parameters, though this had not been well scrutinized in light of simpler explanations.

4.3 Contributions

In this work we consider and empirically investigate several questions surrounding dropout regularization as it applies to feed-forward networks of rectified linear units, the most popular choice at the time. We perform exhaustive experiments in

a variety of simplified conditions in order to gain insights into the use of the geometric mean rather than the arithmetic mean, the accuracy of the weight-scaling approximation to the geometric mean, the regularizing effects of parameter sharing, and the choice of optimizing the log likelihood of individual ensemble members rather than that of the ensemble itself.

4.4 Recent Developments

Dropout continues to be a popular choice for regularization, especially where neural networks are applied to small amounts of labeled data. More recent techniques such as batch normalization (Ioffe and Szegedy, 2015) and residual connections (He et al., 2016) appear to have regularizing effects, which limit the gains available through the additional use of dropout. Continuing work on ensemble methods for neural networks, Hinton et al. (2015) introduced *distillation*, a form of model compression (Buciluǎ et al., 2006) which trains a single network to mimic the outputs of an ensemble of independently trained networks, deriving the benefits of ensembles while limiting computational expense at test time.

5 An Empirical Analysis of Dropout in Piecewise Linear Networks

5.1 Introduction

Dropout (Hinton et al., 2012) has recently garnered much attention as a novel regularization strategy for neural networks involving the use of structured masking noise during stochastic gradient-based optimization. Dropout training can be viewed as a form of ensemble learning similar to bagging (Breiman, 1994) on an ensemble of size exponential in the number of hidden units and input features, where all members of the ensemble share subsets of their parameters. Combining the predictions of this enormous ensemble would ordinarily be prohibitively expensive, but a scaling of the weights admits an approximate computation of the geometric mean of the ensemble predictions.

Dropout has been a crucial ingredient in the winning solution to several high-profile competitions, notably in visual object recognition (Krizhevsky et al., 2012) as well as the Merck Molecular Activity Challenge and the Adzuna Job Salary Prediction competition. It has also inspired work on activation function design (Goodfellow et al., 2013) as well as extensions to the basic dropout technique (Wan et al., 2013; Wang and Manning, 2013) and similar fast approximate model averaging methods (Zeiler and Fergus, 2013).

Several authors have recently investigated the mechanism by which dropout achieves its regularization effect in linear models (Baldi and Sadowski, 2013; Wang and Manning, 2013; Wager et al., 2013), as well as linear and sigmoidal hidden units (Baldi and Sadowski, 2013). However, many of the recent empirical successes of dropout, and feed forward neural networks more generally, have utilised piecewise linear activation functions (Jarrett et al., 2009; Glorot et al., 2011; Goodfellow et al., 2013; Zeiler et al., 2013). In this work, we empirically study dropout in *rectified linear* networks, employing the recently popular hidden unit activation function $f(x) = \max(0, x)$, known as the *rectified linear unit* or *ReLU* activation (Jarrett

et al., 2009; Nair and Hinton, 2010; Glorot et al., 2011). Specifically, we attempt to address the following questions, as they pertain to networks of rectified linear units:

— How accurate is the "weight scaling trick"? How does the use of this approximation impact classification performance?

— Is the geometric mean (approximated by the "weight scaling trick") a suitable replacement for the arithmetic mean, more often employed with ensemble methods?

— What is the role of parameter sharing between ensemble members in dropout, as compared with a conventional ensemble of models with independent parameters?

— Can dropout be adequately explained as regularization via the addition of noise, without appeals to model averaging?

We begin by expanding upon previous work which investigated the quality of dropout's approximate ensemble prediction by comparing against Monte Carlo estimates of the correct geometric average (Srivastava, 2013; Goodfellow et al., 2013). Here, we compare against the true average, in networks of size small enough that the exact computation is tractable. We find, by exhaustive enumeration of all subnetworks in these small cases, that the weight scaling approximation is a remarkably and somewhat surprisingly accurate surrogate for the true geometric mean.

Next, we consider the importance of the geometric mean itself. Traditionally, bagged ensembles produce an averaged prediction via the arithmetic mean, but the weight scaling trick employed with dropout provides an efficient approximation only for the geometric mean. While, as noted by (Baldi and Sadowski, 2013), the difference between the two can be bounded (Cartwright and Field, 1978), it is not immediately obvious what effect this source of error will have on classification performance in practice. We therefore investigate this question empirically and conclude that the geometric mean is indeed a suitable replacement for the arithmetic mean in the context of a dropout-trained ensemble.

The questions raised thus far pertain primarily to the approximate model averaging performed at test time, but *dropout training* also raises some important questions. At each update, the dropout learning rule follows the same gradient that true bagging training would follow. However, in the case of traditional bagging, all members of the ensemble would have independent parameters. In the

case of dropout training, all of the models share subsets of their parameters. It is unclear how much this coordination serves to regularize the eventual ensemble. It is also not clear whether the most important effect is that dropout performs model averaging, or that dropout encourages each individual unit to work well in a variety of contexts.

To investigate this question, we train a set of independent models on resamplings (with replacement) of the training data, as in traditional bagging. Each ensemble member is trained with a single randomly sampled dropout mask fixed throughout all steps of training. We combine these independently trained networks into ensembles of varying size, and compare the ensembles' performance with that of a single network of identical size, trained instead with dropout. We find evidence to support the claim that the weight sharing taking place in the context of dropout (between members of the implicit ensemble) plays an important role in further regularizing the ensemble.

Finally, we investigate an alternative criterion for training the exponentially large shared-parameter ensemble invoked by dropout. Rather than performing stochastic gradient descent on a randomly selected sub-network in a manner similar to bagging, we consider a biased estimator of the gradient of the geometrically averaged ensemble log likelihood (i.e. the gradient of the model being approximately evaluated at test-time), with the particular estimator bearing a resemblance to boosting (Schapire, 1990). We find that this new criterion, employing masking noise with the exact same distribution as is employed by dropout, yields no discernible robustness gains over networks trained with ordinary stochastic gradient descent.

5.2 Review of dropout

Dropout is an ensemble learning and prediction technique that can be applied to deterministic feed-forward architectures that predict a target y given input vector v. These architectures contain a series of hidden layers $\mathbf{h} = \{h^{(1)}, \ldots, h^{(L)}\}$. Dropout trains an ensemble of models consisting of the set of all models that contain a subset of the variables in both v and $\mathbf{h}$. The same set of parameters θ is used to parameterize a family of distributions $p(y \mid v; \theta, \mu)$ where $\mu \in \mathcal{M}$ is a binary mask

vector determining which variables to include in the model, e.g., for a given μ, each input unit and each hidden unit is set to zero if the corresponding element of μ is 0. On each presentation of a training example, we train a different sub-network by following the gradient of $\log p(y \mid v; \theta, \mu)$ for a different randomly sampled μ. For many parameterizations of p (such as most multilayer perceptrons) the instantiation of different sub-networks $p(y \mid v; \theta, \mu)$ can be obtained by elementwise multiplication of v and $\mathbf{h}$ with the mask μ.

5.2.1 Dropout as bagging

Dropout training is similar to bagging (Breiman, 1994) and related ensemble methods (Opitz and Maclin, 1999). Bagging is an ensemble learning technique in which a set of models are trained on different subsets of the same dataset. At test time, the predictions of each of the models are averaged together. The ensemble predictions formed by voting in this manner tend to generalize better than the predictions of the individual models.

Dropout training differs from bagging in three ways:

1. All of the models share parameters. This means that they are no longer really trained on separate subsets of the dataset, and much of what we know about bagging may not apply.
2. Training stops when the ensemble starts to overfit. There is no guarantee that the individual models will be trained to convergence. In fact, typically, the vast majority of sub-networks are never trained for even one gradient step.
3. Because there are too many models to average together explicitly, dropout averages them together with a fast approximation. This approximation is to the geometric mean, rather than the arithmetic mean.

5.2.2 Approximate model averaging

The functional form of the model becomes important when it comes time for the ensemble to make a prediction by averaging together all the sub-networks' predictions. When $p(y \mid v; \theta) = \text{softmax}(v^T W + b)$, the predictive distribution defined by renormalizing the geometric mean of $p(y \mid v; \theta, \mu)$ over $\mathcal{M}$ is simply

given by softmax($v^T W/2 + b$). This is also true for sigmoid output units, which are special cases of the softmax. This result holds exactly in the case of a single layer softmax model (Hinton et al., 2012) or an MLP with no non-linearity applied to each unit (Goodfellow et al., 2013). Previous work on dropout applies the same scheme in deep architectures with hidden units that have nonlinearities, such as rectified linear units, where the $W/2$ method is only an approximation to the geometric mean. The approximation has been characterized mathematically for linear and sigmoid networks (Baldi and Sadowski, 2013; Wager et al., 2013), but seems to perform especially well in practice for nonlinear networks with piecewise linear activation functions (Srivastava, 2013; Goodfellow et al., 2013). We speculate that piecewise linear activation functions allow for an extremely faithful approximation of the model averaging (a claim we empirically verify for rectified linear unit networks in Section 5.4), which in turn boosts generalization performance; the observation of Srivastava (2013) that dropout leads to sparser networks may also play a role (given the ReLU activation's ability to take on a value of exactly zero), although Goodfellow et al. (2013) derived significant benefits from dropout using a non-saturating nonlinearity.

5.3 Experimental setup

Our initial investigations employed rectifier networks with 2 hidden layers and 10 hidden units per layer, and a single logistic sigmoid output unit. We applied this class of networks to six binary classification problems derived from popular multi-class benchmarks, simplified in this fashion in order to allow for much simpler architectures to effectively solve the task, as well as a synthetic task of our own design.

Specifically, we chose four binary sub-tasks from the MNIST handwritten digit database (LeCun et al., 1998). Our training sets consisted of all occurrences of two digit classes (1 vs. 7, 1 vs. 8, 0 vs. 8, and 2 vs. 3) within the first 50,000 examples of the MNIST training set, with the occurrences from the last 10,000 examples held back as a validation set. We used the corresponding occurrences from the official MNIST test set for evaluating test error.

We also chose two binary sub-tasks from the CoverType dataset of the UCI Machine Learning Repository, specifically discriminating classes 1 and 2 (Spruce-Fir vs. Lodgepole Pine) and classes 3 and 4 (Ponderosa Pine vs. Cottonwood/Willow). This task represents a very different domain than the first two datasets, but one where neural network approaches have nonetheless seen success (see e.g. Rifai et al. (2011)). [i]

The final task is a synthetic task in two dimensions: inputs lie in $(-1, 1) \times (-1, 1) \subset \mathbb{R}^2$, and the domain is divided into two regions of equal area: the diamond with corners $(1, 0)$, $(0, 1)$, $(-1, 0)$, $(0, -1)$ and the union of the outlying triangles. In order to keep the synthetic task moderately challenging, the training set size was restricted to 100 points sampled uniformly at random. An additional 500 points were sampled for a validation set and another 1000 as a test set.

In order to keep the mask enumeration tractable in the case of the larger input dimension tasks, we chose to apply dropout in the hidden layers only. This has the added benefit of simplifying the ensemble computation: though dropout is typically applied in the input layer, inclusion probabilities higher than 0.5 are employed (e.g. 0.8 in Hinton et al. (2012); Krizhevsky et al. (2012)), making it necessary to unevenly weight the terms in the average.

Following the general protocol of Goodfellow et al. (2013), we chose hyperparameters by random search (Bergstra and Bengio, 2012) over learning rate and momentum (initial values and decrease/increase schedules, respectively), as well as mini-batch size. We performed early stopping on the validation set, terminating when a lower validation error had not been observed for 100 epochs; when training with dropout, the figure of merit for early stopping was the validation error using the weight-scaled predictions.

We pursued these simplified tasks using small networks in order to perform analyses that involve exhaustive enumeration of the implied ensemble, complementary to Monte Carlo analyses performed on larger networks in the existing literature. In Sections 5.6 and 5.7, more realistic sizes of networks are employed on the full MNIST classification task.

i. Unlike Rifai et al. (2011), we train and evaluate on the records of each class from the data split advertised in the original dataset description. This makes the task much more challenging and many methods prone to overfitting.

5.4 Weight scaling versus Monte Carlo or exact model averaging

Srivastava (2013); Goodfellow et al. (2013) previously investigated the fidelity of the weight scaling approximation in the context of rectifier networks and maxout networks, respectively, through the use of a Monte Carlo approximation to the true model average. By concerning ourselves with small networks where exhaustive enumeration is possible, we were able to avoid the effect of additional variance due to the Monte-Carlo average and compute the exact geometric mean over all possible dropout sub-networks.

On each of the 7 tasks, we randomly sampled 50 sets of hyperparameters and trained 50 networks with dropout. We then computed, for each point in the test set for each task, the activities of the network corresponding to each of the 2^{20} possible dropout masks. We then geometrically averaged their predictions (by arithmetically averaging all values of the input to the sigmoid output unit) and computed the geometric average prediction for each point in the test set. Finally, we compared the misclassification rate using these predictions to that obtained using the approximate, weight-scaled predictions.

The results are shown in Figure 5.1, where each point represents a different hyperparameter configuration. The overall result is that the approximation yields a network that performs remarkably and surprisingly similarly. We statistically tested the fidelity of the approximation via the Wilcoxon signed-rank test (Wilcoxon, 1945), a nonparametric paired sample test similar to the paired t-test, applying a Bonferroni correction (Abdi, 2007) for multiple hypotheses (i.e. dividing the null rejection threshold α by the number of tests performed). At $\alpha = 0.01$, no significant differences were observed for any of the seven tasks.

5.5 Geometric mean versus arithmetic mean

Though the inexpensive computation of an approximate geometric mean was noted in (Hinton et al., 2012), little has been said of the choice of the geometric mean. Ensemble methods in the literature often employ an arithmetic mean

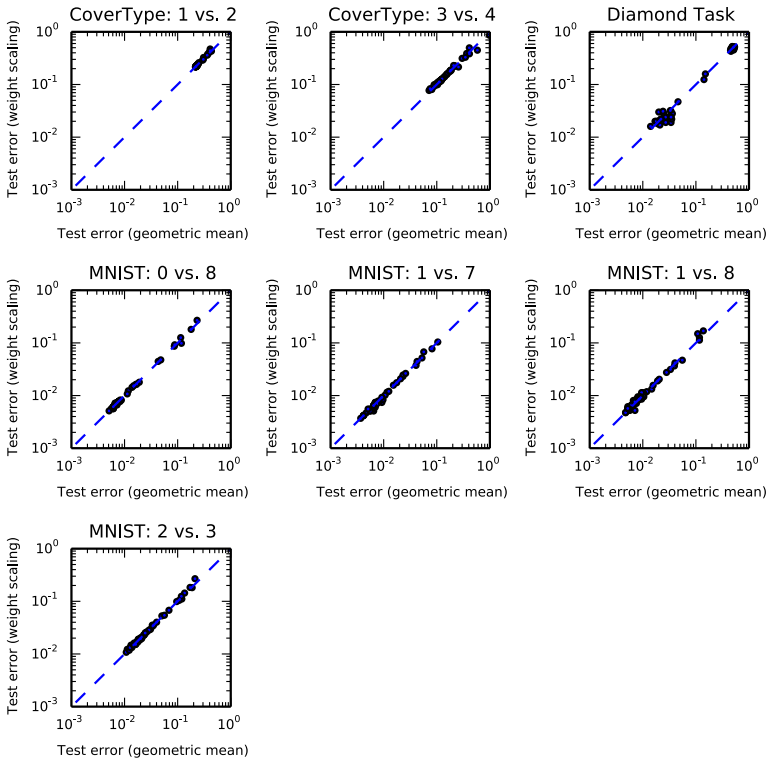

Figure 5.1 – Comparison of test error obtained with an exhaustive computation of the geometric mean (x-axis) and the weight-scaling approximation (y-axis). Each point represents a network trained with different hyperparameters.

for model averaging. It is thus natural to pose the question as to whether the choice of the geometric mean has an impact on the generalization capabilities of the ensemble. Computing the exact arithmetic mean prediction carries a similar computational cost to exact computation of the geometric mean prediction.

Using the same networks trained in Section 5.4, we combined the forward-propagated predictions of all 2^{20} models using the arithmetic mean. In Figure 5.2, we plot the test error using the (exact) arithmetic mean prediction against that obtained using the (exact) geometric mean prediction. We find that across all seven tasks, the geometric mean is a reasonable proxy for the arithmetic mean, with relative error rarely exceeding 20% except for the synthetic task. In absolute terms, the discrepancy between the test error achieved by the geometric mean and the arithmetic mean never exceeded 0.75% for any of the tasks. Importantly, we find no evidence of a systematic bias in favour of the arithmetic mean.

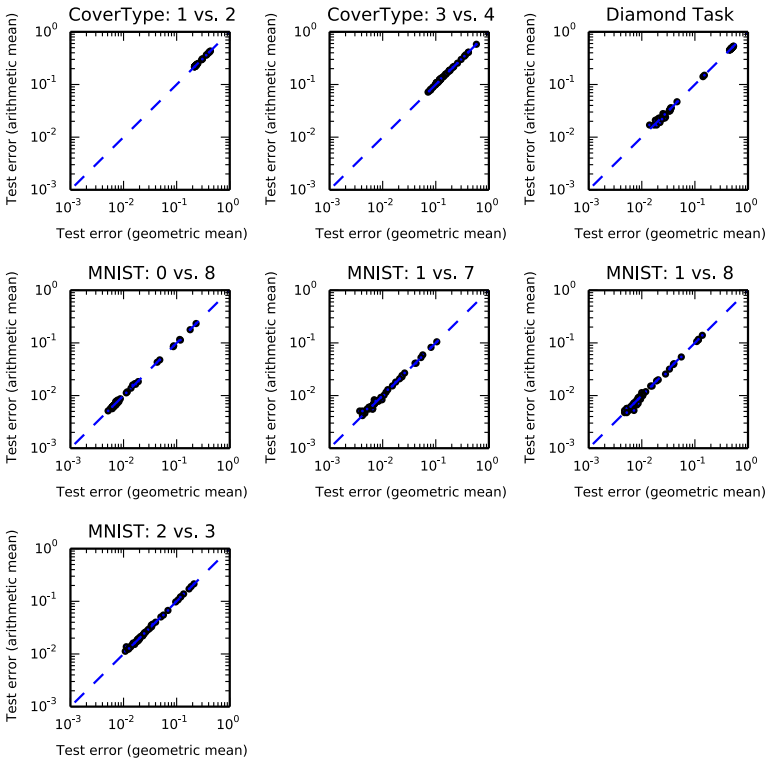

Figure 5.2 – Comparison of test error obtained with an exhaustive computation of the arithmetic mean (x-axis) geometric mean (y-axis). Each point represents a network trained with different hyperparameters.

5.6 Dropout ensembles versus untied weights

We now turn from our investigation of the characteristics of inference in dropout-trained networks to an investigation of the training procedure. For the remainder of the experiments discussed, we trained networks of a more realistic size and capacity on the full multiclass MNIST problem. Once again, we employed two layers of rectified linear units. In addition to dropout, we utilised norm constraint regularization on the incoming weights to each hidden unit (Srebro and Shraibman, 2005; Srivastava, 2013). We again performed random search over hyperparameter values, now including in our search the initial ranges of weights, the number of hidden units in each of two layers, and the maximum weight vector norms of each layer.

Dropout training can be viewed as performing bagging on an ensemble that is of

size exponential in the number of hidden units, where each member of the ensemble shares parameters with other members of the ensemble. Because each gradient step is taken on a different mini-batch of training data, each sub-network can be seen to be trained on a different resampling of the training set, as in traditional bagging. Furthermore, while each step is taken with respect to the log likelihood of a single ensemble member, the effect of the weight update is applied to all members of the ensemble simultaneously.[i] We investigate the role of this complex weight-sharing scheme by training an ensemble of independent networks on resamplings of the training data, each with a single dropout mask fixed in place throughout training.

We first performed a hyperparameter search by sampling 50 hyperparameter configurations and choosing the network with the lowest validation error. The best of these networks obtains a test error of 1.06%, matching results reported by Srivastava (2013). We then trained 600 models initialized with different random seeds, on different resamplings (with replacement) of the training set, as in traditional bagging. Instead of applying dropout during training (and thus applying a different mask at each gradient step), we sampled one dropout mask per ensemble member and held it fixed throughout training and at test time. In order to facilitate a fairer comparison while still matching the capacity (on average) of individual ensemble members, we fixed the number of hidden units in each layer ($2,834$ in layer 1, $3,219$ in layer 2) to be equal to the layer dimensions in the best performing dropout network. However, we then re-ran random search over the remaining hyperparameters *without* dropout, once again sampling 50 configurations, in order to determine the hyperparameters for the fixed-mask ensemble members.[ii]

The resulting networks thus have architectures sampled from the same distribution as the sub-networks trained during the best run of dropout training, but each network's parameters are independent of all other networks. This ensemble necessarily comes at considerably higher computational expense than a single network trained with dropout. Note that each member of the ensemble was individually early-stopped.

i. At least, all members of the ensemble that share any parameters with the sub-network just updated. There certainly exist pairs of ensemble members whose parameter sets are disjoint.

ii. The motivation for this protocol was to control for layer size while noting that the optimal hyperparameters for training individual members of the untied ensemble may be substantially different the optimal hyperparameters for training with dropout, and might more closely resemble the optimal hyperparameters for training a network without dropout and without any mask.

We evaluated test error for ensembles of these networks, combining their predictions (with the dropout mask used during training still fixed in place at test time) via the geometric mean, as is approximately done in the context of dropout. Our results for various sizes of ensemble are shown in Figure 5.3. Our results

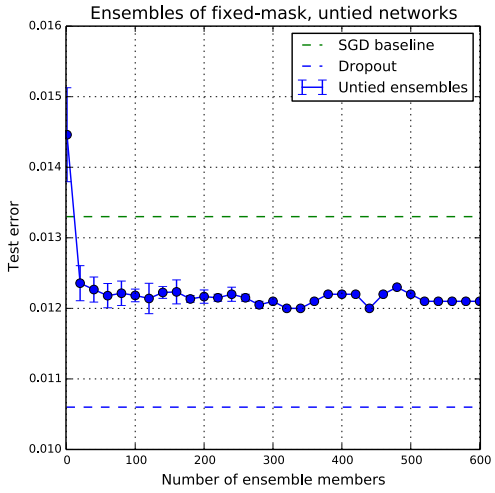

Figure 5.3 – Average test error on MNIST for varying sizes of untied-weight ensembles. 600 networks were trained to convergence, each with a single randomly sampled dropout mask fixed in place throughout. These networks' pre-softmax activations were then averaged to produce predictions for varying sizes of ensembles. For each size n, $\lfloor 600/n \rfloor$ disjoint subsets were combined in this fashion, and the test error mean and standard deviation over ensembles is shown here.

suggest that while bagging an ensemble of dropped-out networks improves generalization performance over the single network of equivalent size trained with SGD, the gains from larger ensembles appear to quickly diminish, and dropout nonetheless performs considerably better. This suggests that parameter sharing amongst subnetworks in the dropout ensemble plays a significant role in regularizing the resulting network. These results do rely on relatively small ensembles, however, with the same hyperparameters used for each ensemble member; it remains unclear how to efficiently optimize hyperparameters for the individual members of a large ensemble so as to facilitate an even fairer comparison (this highlights a general issue with the high cost of training ensembles of neural networks, that dropout conveniently sidesteps).

5.7 Dropout bagging versus dropout boosting

Other algorithms such as denoising autoencoders (Vincent et al., 2010) are motivated by the idea that models trained with noise are robust to slight transformations of their inputs. Previous work has drawn connections between noise and regularization penalties (Bishop, 1995); similar connections in the case of dropout have recently been noted (Baldi and Sadowski, 2013; Wager et al., 2013). It is natural to question whether dropout can be wholly characterized in terms of learned noise robustness, and whether the model-averaging perspective is necessary or fruitful.

In order to investigate this question we propose an algorithm that injects exactly the same noise as dropout. For this test to be effective, we require an algorithm that can successfully minimize training error, and obtain acceptable generalization performance. It needs to perform at least as well as standard maximum likelihood; otherwise all we have done is designed a pathological algorithm that fails to train.

We therefore introduce *dropout boosting*. The objective function for each (sub-network, example) pair in dropout boosting is the likelihood of the data according to the ensemble; however, only the parameters of the current sub-network may be updated for each example. Ordinary dropout performs bagging by maximizing the likelihood of the correct target for the current example *under the current sub-network*, whereas dropout boosting takes into account the contributions of other sub-networks, in a manner reminiscent of boosting.

The objective function for dropout is $\frac{1}{2^{|\mathcal{M}|}} \sum_{\mu \in \mathcal{M}} \log p(y \mid v; \theta, \mu)$. For dropout boosting, assume each mask μ has a separate set of parameters θ_μ (though in reality these parameters are tied, as in conventional dropout). The dropout boosting objective function is then given by $\log p_{\text{ensemble}}(y \mid v; \theta)$, where

$$p_{\text{ensemble}}(y \mid v; \theta) = \frac{1}{Z} \tilde{p}(y \mid v; \theta)$$

$$Z = \sum_{y'} \tilde{p}(y' \mid v; \theta)$$

$$\tilde{p}(y \mid v; \theta) = \sqrt[2^{|\mathcal{M}|}]{\prod_{\mu \in \mathcal{M}} p(y \mid v; \theta_\mu)}.$$

The boosting learning rule is to select one model and update its parameters given all of the other models. In conventional boosting, these other models have already

been trained to convergence. In dropout boosting, the other models actually share parameters with the network being trained at any given step, and initially the other models have not been trained at all. The learning rule is to select a sub-network indexed by μ and follow the ensemble gradient $\nabla_{\theta_\mu} \log p_{\text{ensemble}}(y \mid v; \theta)$, i.e.

$$\Delta\theta_\mu \propto \frac{1}{2^{|\mathcal{M}|}} \left(\nabla_{\theta_\mu} \log p(y \mid v; \theta_\mu, \mu) + \sum_{y'} p_{\text{ensemble}}(y' \mid v) \nabla_{\theta_\mu} \log p(y' \mid v; \theta_\mu, \mu) \right).$$

Rather than using the boosting-like algorithm, one could obtain a generic Monte-Carlo procedure for maximizing the log likelihood of the ensemble by averaging together the gradient for multiple values of μ, and optionally using a different μ for the term in the left and the term on the right. Empirically, we obtained the best results in the special case of boosting, where the term on the left uses the same μ as the term on the right – that is, both terms of the gradient apply updates only to one member of the ensemble, even though the criterion being optimized is global.

Note that the intractable p_{ensemble} still appears in the learning rule. To implement the training algorithm efficiently, we can approximate the ensemble predictions using the weight scaling approximation. This introduces further bias into the estimator, but our findings in Section 5.4 suggest that the approximation error is small.

Note that dropout boosting employs exactly the same noise as regular dropout uses to perform bagging. Indeed, the first term of the dropout boosting update is simply the update utilized by dropout. The second term is the gradient of the log likelihood of the same randomly chosen sub-network (i.e. same dropout mask) but substituting the true targets with the approximately-averaged ensemble prediction. Training proceeds in the same fashion as dropout, where one randomly selected sub-network (from the same distribution over masks) is updated, but according to a more globally aware criterion. If the mask application is viewed merely as the addition of noise, both criteria employ identical noise, as the selection procedure and random distribution over masks is identical. Dropout boosting should thus perform similarly to conventional dropout if learned noise robustness is the important ingredient.

If we instead take the view that this is a large ensemble of complex learners

whose likelihood is being jointly optimized, we would expect that employing a criterion more similar to boosting than bagging would perform more poorly. As boosting maximizes the likelihood of the ensemble, it would perhaps be prone to overfitting in this setting, as the ensemble is very large and the learners are not particularly weak.

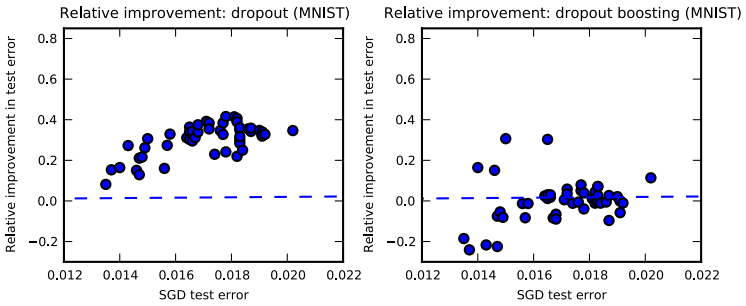

Figure 5.4 – Comparison of dropout (left) and dropout boosting (right) to stochastic gradient descent with matched hyperparameters.

Starting with the 50 models trained in Section 5.6, we employed the same hyperparameters to train a matched set of 50 networks with dropout boosting, and another with plain stochastic gradient descent. In Figure 5.4, we plot the relative performance of dropout and dropout boosting compared to a model with the same hyperparameters trained with SGD. While dropout unsurprisingly shows a very consistent edge, dropout boosting performs, on average, little better than stochastic gradient descent. The Wilcoxon signed-rank test similarly failed to find a significant difference between dropout boosting and SGD ($p > 0.7$). While several outliers approach very good performance (perhaps owing to the added stochasticity), dropout boosting is, on average, no better and often slightly worse than maximum likelihood training, in stark contrast with dropout's systematic advantage in generalization performance.

5.8 Conclusion

We investigated several questions related to the efficacy of dropout, focusing on the specific case of the popular rectified linear nonlinearity for hidden units. We showed that the weight-scaling approximation is a remarkably accurate proxy

for the usually intractable geometric mean over all possible sub-networks, and that the geometric mean (and thus its weight-scaled surrogate) compares favourably to the traditionally popular arithmetic mean in terms of classification performance. We demonstrated that weight-sharing between members of the implicit dropout ensemble appears to have a significant regularization effect, by comparing to analogously trained ensembles of the same form that did not share parameters. Finally, we demonstrated that simply adding noise, even noise with identical characteristics to the noise applied during dropout training, is not sufficient to obtain the benefits of dropout, by introducing dropout boosting, a training procedure utilising the same masking noise as conventional dropout, which successfully trains networks but loses dropout's benefits, instead performing roughly as well as ordinary stochastic gradient descent.

Our results suggest that dropout is an extremely effective ensemble learning method, paired with a clever approximate inference scheme that is remarkably accurate in the case of rectified linear networks. Further research is necessary to shed more light on the model averaging interpretation of dropout. Hinton et al. (2012) noted that dropout forces each hidden unit to perform computation that is useful in a wide variety of contexts. Our results with a sizeable ensemble of independent bagged models seem to lend support to this view, though our experiments were limited to ensembles of several hundred networks at most, tiny in comparison with the weight-sharing ensemble invoked by dropout. The relative importance of the astronomically large ensemble versus the learned "mixability" of hidden units remains an open question. Another interesting direction involves methods that are able to efficiently, approximately average over different classes of model that share parameters in some manner, rather than merely averaging over members of the same model class.

Acknowledgments

The authors would like to acknowledge the efforts of the many developers of Theano (Bergstra et al., 2010; Bastien et al., 2012), pylearn2 (Goodfellow et al., 2013) which were utilised in experiments. We would also like to thank NSERC, Compute Canada, and Calcul Québec for providing computational resources. Ian Goodfellow is supported by the 2013 Google Fellowship in Deep Learning.

6 Prologue to Third Article

6.1 Article Details

Self-informed neural network structure learning. D. Warde-Farley, A. Rabinovich, D. Anguelov. *Proceedings of the 30th International Conference on Learning Representations (ICLR '15)*, Workshop Track.

Personal Contribution. The idea for training a convolutional network classifier with dedicated capacity informed by structure imposed on the label space is primarily attributable to Andrew Rabinovich. The idea of the specific variant that involved preserving "generalist" capacity in the final portion of the network was my own. I implemented the procedure in Google's asynchronous distributed neural networks platform DistBelief (Dean et al., 2012). I trained all of the augmented models reported in this work, and wrote the manuscript. Andrew Rabinovich provided his implementation of the spectral clustering method, and evaluated the resultant models on the ImageNet benchmark. I devised the control experiment.

6.2 Context

At the time that I interned with the Image Understanding team at Google in the summer of 2014, they had recently adopted the Inception (Szegedy et al., 2015) family of convolutional network image classifiers for use in the Photo Search product. These classifiers would process user-stored photos and annotate them with metadata about object classes detected in each, in order that users may search their photos via textual queries. The Inception architecture was quite exotic, and carefully tuned to the point that it was very difficult to improve upon: most modifications to the architecture were deleterious. Furthermore, these classifiers were

trained using asynchronous stochastic gradient descent for months, making it computationally costly to experimentally iterate; most training of hypothesized improvements proceeded from the checkpointed parameters of a similar, previously trained architecture. It was clear that improvements ought to be possible, as certain confusions were systematic.

6.3 Contributions

The contribution of this work is that of one successful attempt at addressing the problem of improving upon a high-performing neural network architecture by carefully augmenting it with pathways dedicated to groups of labels that are more frequently confused by the unaugmented architecture.

6.4 Recent Developments

As of this writing, design of new neural network architectures remains more of an art than a science. Progress has been made in understanding the unreasonable effectiveness of the popular Inception architecture (Szegedy et al., 2015), while recent work (He et al., 2016) has reintroduced the idea of "shortcut" or "skip-layer" connections (Schraudolph, 1998) into modern convolutional networks and allowing for the training of networks hundreds of layers deep. Key to the success of these residual networks, or ResNets, is *batch normalization* (Ioffe and Szegedy, 2015), a training acceleration technique introduced shortly after this work was completed. Finally, very recent work (Zoph et al., 2017) has taken the first steps towards automated discovery of superior convolutional architectures.

7 Self-Informed Neural Network Structure Learning

7.1 Introduction

In the context of large scale visual recognition, it is not uncommon for state-of-the-art convolutional networks to be trained for days or weeks before convergence (Krizhevsky et al., 2012; Sermanet et al., 2014; Szegedy et al., 2015). Performing exhaustive architecture search is quite challenging and computationally expensive. Furthermore, once a satisfactory architecture has been discovered, it can be extremely difficult to improve upon; small changes to the architecture more often decrease performance than improve it. In architectures containing fully-connected layers, naively increasing the dimensionality of such layers increases the number of parameters between them quadratically, increasing both the computational workload and the tendency towards overfitting.

In settings where the domain of interest comprises thousands of classes, improving performance on specific subdomains can prove challenging, as the jointly learned features that succeed on the overall task on average may not be sufficient for correctly identifying the "long tail" of classes, or for making fine-grained distinctions between very similar entities. Side information in the form of metadata – for example, from Freebase (Bollacker et al., 2008) – often only roughly corresponds to the kind of similarity that would make correct classification challenging. In the context of object classification, visually similar entities may belong to vastly different high-level categories (e.g. a sporting activity and the equipment used to perform it), whereas two entities in the same high-level semantic category may bear little resemblance to one another visually.

A traditional approach to building increasingly accurate classifiers is to average the predictions of a large ensemble. In the case of neural networks, a common approach is to add more layers or making existing layers significantly larger, possibly with additional regularization. These strategies present a significant problem

in runtime-sensitive production environments, where a classifier must be rapidly evaluated in a matter of milliseconds to comply with service-level agreements. It is therefore often desirable to increase a classifier's capacity in a way that significantly improves performance while minimally impacting the computational resources required to evaluate the classifier; however, it is not immediately obvious how to satisfy these two competing objectives.

We present a method for judiciously adding capacity to a trained neural network using the network's own predictions on held-out data to inform the augmentation of the network's structure. We demonstrate the efficacy of this method by using it to significantly improve upon the performance of a state-of-the-art industrial object recognition pipeline based on Szegedy et al. (2015) with less than 3% extra computational overhead.

7.2 Methods

Given a trained network, we evaluate the network on a held out dataset in order to compute a confusion matrix. We then apply spectral clustering (Chung, 1997) to generate a partitioning of the possible labels.

We augment the trained network's structure by adding additional stacks of fully connected layers, connected in parallel with the pre-existing stack of fully-connected layers. The output of each "auxiliary head" is connected by a weight matrix only to a subset of the output units, corresponding to the label clusters discovered by spectral clustering.

We train the augmented network by initializing the pre-existing portions of the network (minus the classifier layer's weights and biases) to the parameters of the original network, and by randomly initializing the remaining portions. We train holding the pre-existing weights and biases fixed, learning only the hidden layer weights for the new portions and retraining the classifier layer's weights. This allows for training to focus on making good use of the auxiliary capacity rather than adapting the pre-initialized weights to compensate for the presence of the new hidden units. Note that it is also possible to fine-tune the whole network after training the augmented section, though we did not perform such fine-tuning in the experiments described below.

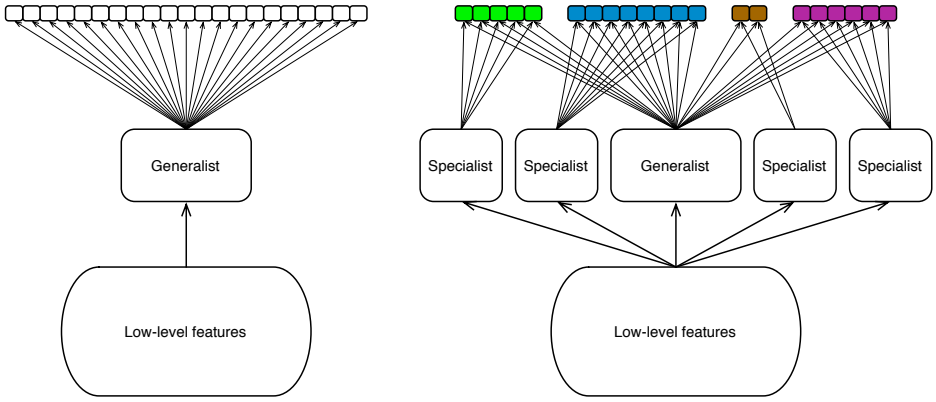

Figure 7.1 – A schematic of the augmentation process. Left: the original network. Right: the network after augmentation.

7.3 Related work

Our method can be seen as similar in spirit to the mixture of experts approach of Jacobs et al. (1991). Rather than jointly learning a gating function as well as experts to be gated, we employ as a starting point a strong generalist network, whose outputs then inform decisions about which specialist networks to deploy for different subsets of classes. Our specialists also do not train with the original data as input but rather a higher-level feature representation output by the original network's convolutional layers.

Recent work on distillation (Hinton et al., 2014), building on earlier work termed model compression (Buciluǎ et al., 2006), emphasizes the idea that a great deal of valuable information can be gleaned from the non-maximal predictions of neural network classifiers. Distillation makes use of the averaged overall predictions of several expensive-to-evaluate neural networks as "soft targets" in order to train a single network to both predict the correct label and mimic the overall predictions of the ensemble as closely as possible. As in Hinton et al. (2014), we use the predictions of the model itself, however we use this knowledge in the pursuit of carefully adding capacity to a single, already trained network, rather than mimicking the performance of many networks with one. Our approach is arguably complementary, and could conceivably be applied after distilling an ensemble into a single mimic network in order to further improve fine-grained performance.

7.4 Experiments

Our base model consists of the same convolutional Inception architecture employed in GoogLeNet (Szegedy et al., 2015), plus two fully connected hidden layers of 4,096 rectified linear (ReLU) units each. Our output layer consists of logistic units, one per class.

We evaluated the trained network on 9 million images not used during training. Let

$$g_j(x) = \begin{cases} 1, & \text{if example } x \text{ has ground truth annotation for class } j \\ 0, & \text{otherwise} \end{cases} \tag{7.1}$$

$$M_{i,K}(x) = \begin{cases} 1, & \text{if model } M\text{'s top } K \text{ predicted labels on example } x \text{ includes class } i \\ 0, & \text{otherwise} \end{cases} \tag{7.2}$$

We compute the following matrix on the hold-out set S:

$$A = [a_{ij}]; \quad a_{ij} = \mathbb{E}_{x \in S} \left[M_{i,K}(x) \cdot g_j(x) \right] \tag{7.3}$$

using $K = 100$. We use the seemingly large value of $K = 100$ in order to recover annotations for a large fraction of possible classes on at least one example in the hold-out set. We term the detection of class i in the context of ground truth class j a *confusion* of i with j; the (i, j)th entry of this matrix thus encodes the fraction of the time class i is "confused" with class j on the hold-out set.

We also experimented with the matrix

$$A = [a_{ij}]; \quad a_{ij} = \mathbb{E}_{x \in S} \left[M_{i,K}(x) \cdot M_{j,K}(x) \right] \tag{7.4}$$

wherein we eschew the use of ground truth and only look at *co-detections*, again with $K = 100$.

We symmetrize either matrix as $B = A^{\mathsf{T}} A$, and apply spectral clustering using B as our similarity matrix, following the formulation of Ng et al. (2002). In all of our experiments, our specialist sub-networks consisted of two layers of 512 ReLUs each.

We evaluate our method on an expanded version of the JFT dataset described

in Hinton et al. (2014), an internal Google dataset with a training set of approximately 100 million images spanning 17,000 classes.

7.5 Results

7.5.1 Label clusters recovered

In Table 7.1, we observe that spectral clustering on the matrix B was able to successfully recover clusters consisting of visually similar entities.

Runway, Handshake, Douglas dc-3, Tarmac, Boeing, Air show, Interceptor, Hospital ship, Coast guard, Republic p-47 thunderbolt, Sikorsky sh-3 sea king, Boeing 737, Mcdonnell douglas dc-10, Air force, Boeing 757, Boeing 717, Hovercraft, Lockheed ac-130, McDonnell Douglas, Travel, Aircraft engine, Flight, Yawl, Lockheed c-5 galaxy, Cockpit, Bomber, Lockheed p-3 orion, Avro lancaster, Jet aircraft...
Pickled food, Grilled food, North african cuisine, Vinegret, Woku, Lasagne, Lard, Meringue, Peanut butter and jelly sandwich, Sparkling wine, Salting, Raclette, Mussel, Galliformes, Chemical compound, Succotash, Cucurbita, Alcoholic beverage, Bento, Osechi, Okonomiyaki, Nabemono, Miso soup, Dango, Onigiri, Tempura, Mochi, Soba, Shiitake, Indian cuisine, Andhra food, Foie gras, Krill, Sour cream, Saumagen, Compote...
Lingonberry, Rooibos, Persimmon, Rutabaga, Banana family, Ensete, Apple, Viola, Shamrock, Walnut, Beech, Poppy, Kimjongilia, Chicory, Bay leaf, Melon, Grain, Juniper, Spruce, Fir, Birch family, Hawthorn, Guava, Gooseberry, Tick, Pouchong, Bonsai, Caraway, Fennel, Sea anemone, Maple sugar, Boysenberry, Mustard and cabbage family, Pond, Moss, Daikon, Wild ginger, Groundcover, Holly, Viburnum lentago, Ivy family, Mustard seed...

Table 7.1 – Examples of partial sets of labels grouped together by performing spectral clustering on the base network's confusions, based on the 100 top scoring predictions. The first row appears aviation-related, the second focusing on mainly food, and the third broadly concerned with plant-related entities.

7.5.2 Test set performance improvements

We evaluate on a balanced test set with the same number of classes per image. For each of the confusion and co-detection cases, we compare against a network with identical capacity and topology (i.e. same number of labels per cluster) with labels randomly permuted, in order to assess the importance of the particular partitioning discovered while carefully controlling for the number of parameters being learned.

Description	mAP @ top 50	# Multiply-Adds	Extra Computation
Base network	36.80%	1.52B	1.000×
Base + 6 heads, confusions	39.41%	1.56B	1.026×
Base + 6 heads, randomized	32.97%	”	”
Base + 13 heads, co-detections	38.07%	1.60B	1.053×
Base + 13 heads, randomized	32.13%	”	”

Table 7.2 – Summary of the performance of augmented networks and the extra computation time incurred.

While both methods improve upon the base network, the use of ground truth appears to provide a significant edge. Our best performing network, with 6 specialist heads, increases the number of multiply-adds required for evaluation from 1.52 billion to 1.56 billion, a modest increase of 2.6%.

We also provide, in Figure 7.2, an evaluation of our best performing JFT network against the ImageNet 1,000-class test set, on the subset of JFT classes that can be mapped to classes from the ImageNet task (approximately 660 classes). These results are thus not directly comparable to results obtained on the ImageNet training set; a more direct comparison is left to follow-up work.

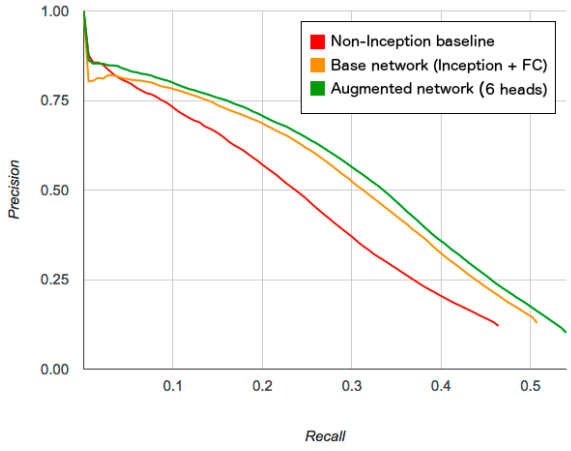

Figure 7.2 – A preliminary evaluation of our trained network on the subset of classes in JFT that are mappable to the 1,000-class ImageNet classification task.

7.6 Conclusions & Future Work

We have presented a simple and general method for improving upon trained neural network classifiers by carefully adding capacity to groups of output classes that the trained model itself considers similar. While we demonstrate results on a computer vision task, this is not an assumption underlying the approach, and we plan to extend it to other domains in follow-up work.

In these experiments we have allocated a fixed extra capacity to each label group, regardless of the number of labels in that group. Further investigation is needed into strategies for the allocation of capacity to each label group. Seemingly relevant factors include both the cardinality of each group and the amount of training data available for the labels contained therein; however, the difficulty of the discrimination task does not necessarily scale with either of these.

In the case of the particular convolutional network we have described, it is not obvious that the best place to connect these auxiliary stacks of hidden layers is following the last convolutional layer. Most of the capacity, and therefore arguably most of the discriminative knowledge in the network, is contained in the fully connected layers, and appealing to this part of the network for augmentation purposes seems natural. Nonetheless, it is possible that one or more layers of group-specific convolutional feature maps could be beneficial as well. Note that the augmentation procedure could also theoretically be applied more than once, and not necessarily in the same location. Each subsequent clustering and retraining step could potentially identify a complementary division of the label space, capturing new information.

Finally, this can be seen as a small step towards the "conditional computation" envisioned by Bengio (2013), wherein relevant pathways of a large network are conditionally activated based on task relevance. Here we have focused on the relatively large gains to be had with computationally inexpensive, targeted augmentations. Similar strategies could pave the way towards networks with much higher capacity specialists that are only evaluated when necessary.

8 Adversarial Networks

In this chapter, we review the recently introduced generative adversarial network (GAN) paradigm (Goodfellow et al., 2014). A modified and extended version of this text was published in Warde-Farley and Goodfellow (2016).

Generative adversarial networks phrase the problem of estimating a generative model in terms of a sample generation process $G : \mathbb{R}^d \to \mathbb{R}^n$, which takes as its argument a random variate $\mathbf{z} \sim p(\mathbf{z})$; $p(\mathbf{z})$ is often chosen from some simple family such as an isotropic Gaussian distribution, or a uniform distribution on $[-1, 1]^d$. $G(\cdot)$ is a machine parameterized by Θ_G which learns to map a sample from the base distribution $p(\mathbf{z})$ to a corresponding sample from an implicitly defined distribution $p_g(\mathbf{x})$. The combined procedure of drawing a sample $\mathbf{z}$ from $p(\mathbf{z})$ and applying G to $\mathbf{z}$ is referred to as the *generator*.

In contrast with many existing generative modeling frameworks, GANs may be trained without an explicit algebraic representation of $p_{\text{model}}(x)$, tractable or otherwise. The GAN framework is compatible with some models that explicitly define a probability distribution—any directed graphical model whose sampling process is compatible with stochastic back-propagation (Williams, 1992; Kingma and Welling, 2014; Rezende et al., 2014) may be used as a GAN generator—but the framework does not require explicit specification of any conditional or marginal distributions, only the sample generation process. In frameworks based on explicit specification of probabilities it is typical to maximize the empirical expectation of $\log p_{\text{model}}(\mathbf{x})$, applying Monte Carlo or variational approximations if faced with intractable terms (often in the form of a normalizing constant). Instead, GANs are trained to match the data distribution indirectly with the help of a *discriminator*, i.e. a binary classifier $D : \mathbb{R}^n \to [0, 1]$, parameterized by Θ_D, whose output represents a calibrated probability estimate that a given example was sampled from $p_{\text{data}}(\mathbf{x})$. The conditional log likelihood of the discriminator, on a balanced dataset of real and synthetic examples, is (in the usual fashion) *maximized* with respect to the parameters of D, but simultaneously *minimized* with respect to the parameters

of G.

8.1 Adversarial networks in theory and practice

The joint training procedure for the generator G and the discriminator D can be viewed as a two-player, continuous minimax game with a certain value function. In their introduction of the GAN framework, Goodfellow et al. (2014) proved that the GAN training criterion has a unique global optimum in the space of distributions represented by G and D, wherein the distribution sampled by the generator exactly matches that of the data generating process, and the discriminator D is completely unable to distinguish real data from synthetic. It can also be proved, under certain assumptions, that the game converges to this optimum if G is improved at every round and D is chosen to be the ideal discriminator between $p_g(\mathbf{x})$ and $p_{\text{data}}(\mathbf{x})$, i.e. $D^\star(\mathbf{x}) = p_{\text{data}}(\mathbf{x})/(p_{\text{data}}(\mathbf{x}) + p_g(\mathbf{x}))$.

Goodfellow (2014) advanced the theoretical understanding of the GAN training criterion and its relationship to other distinguishability-based learning criteria. In particular, *noise-contrastive estimation (NCE)* (Gutmann and Hyvarinen, 2010) can be viewed as a variant of the GAN criterion wherein the generator is fixed, and the discriminator is a generatively parameterized classifier that learns an explicit model of $p(\mathbf{x})$ as a side effect of discriminative training, while a variant of noise contrastive estimation employing (a copy of) the learned generative model is shown to be equivalent, in expectation, to maximum likelihood. Perhaps most importantly, Goodfellow (2014) noted a subtlety of theoretical results outlined above, pointing out that they are significantly weakened by the setting in which GANs are typically optimized in practice.

Optimization of the generator and discriminator necessarily takes place in the space of parameterized families of functions, and the cost surface in the space of these parameters may have symmetries and other pathologies that imply non-uniqueness of the optima as well as practical difficulties locating them. One does not typically have analytical access to $p_g(\mathbf{x})$ and certainly not to $p_{\text{data}}(\mathbf{x})$, and must attempt to infer the optimal discriminator from data and samples. It is often prohibitively expensive to fully optimize the parameters of D after every change in the parameters of G – therefore, in practice, one settles for a parameter update

aimed at improving D, such as one or more stochastic gradient steps. This means that the generator's role in the minimax game of minimizing with respect to $p_g(\mathbf{x})$ given a *maximum* of the value function with respect to D, is instead *minimizing a lower bound* on the correct objective. It is not at all clear whether the minimization of this lower bound improves the quantity of interest or simply loosens the bound.

Note that Goodfellow et al. (2014) optimize a slightly different criterion than described above. Let $D(x) = p(\mathbf{x} \text{ is data} \mid \mathbf{x})$, the discriminator's estimate that a given sample $\mathbf{x}$ comes from the data. Rather than minimize

$$\mathbb{E}_{\mathbf{z}\sim p(\mathbf{z})} \log\left(1 - D(G(\mathbf{z}))\right)$$

(a term that already appears in the training criterion for the discriminator) with respect to the parameters of G, one can instead maximize $\mathbb{E}_{\mathbf{z}\sim p(\mathbf{z})} \log\left(D(G(\mathbf{z}))\right)$; this criterion was found to work better in practice. The motivation for this lies in the fact that early in training, when G is producing samples that look nothing at all like data, the discriminator D can quickly learn to distinguish the two and $\log\left(1 - D(G(\mathbf{z}))\right)$ can quickly saturate to zero. The derivative of the per-sample objective contains a factor of $\left(1 - D(G(\mathbf{z}))\right)^{-1}$, thus scaling the gradients which G receives via backpropagation to have very small magnitude. Pushing upward on $\log D(G(\mathbf{z}))$ yields a multiplicative factor of $D(G(\mathbf{z}))^{-1}$ instead, resulting in gradients with a more favourably scaled magnitude if $D(G(\mathbf{z}))$ is small.

As G and D are both parameterized learners, the balance between the respective modeling capacities (and effective capacities during learning) can have a profound effect on the learning dynamics and the success of generative learning. In particular, the discriminator must be sufficiently flexible to reliably model the difference between the data distribution and the generated distribution, as the latter gradually tends towards reproducing the statistical structure of the former. At the same time, the discriminator must not become too effective too quickly, or else the gradients it provides the generator will be uninformative: no small change in the generated sample will move it significantly closer to the discriminator's decision boundary.

8.2 Generator collapses

Note that in theory, a perfectly optimal discriminator could exploit any subtle mismatch between $p_{\text{data}}(\mathbf{x})$ and $p_g(\mathbf{x})$ to give itself a better-than-chance ability to correctly distinguish real and synthetic examples; the generator could then use the gradients obtained from this optimal discriminator to correct its misallocations of probability mass. In practice, when using richly parameterized neural networks for generation and discrimination, the objective functions used to train the generator are non-convex and (due to the dependence between the learning tasks for the generator and the discriminator) highly nonstationary; it is impractical and even theoretically intractable to globally optimize the discriminator prior to each change in the generator. A failure mode for the training criterion therefore manifests when the generator learns to place too much probability mass on a subregion of the data distribution. In the most extreme cases, a generator could elect to place all of its mass on a single point, perfectly reproducing a single training example. A well-trained discriminator can quickly learn to exploit this and confidently classify every other point in the training set correctly. This presents a problem for generator learning, in that the gradients the generator receives are entirely with respect to a single synthetic example, most local perturbations of which will result in gradients that point back towards the singularity. To date, strategies to mitigate this type of failure are an active area of research. Radford et al. (2015) noted that the judicious use of batch normalization (Ioffe and Szegedy, 2015) appears, empirically, to prevent these kinds of collapses to a large degree.

8.3 Sample fidelity and learning the objective function

Machine learning problems are classically posed in terms of an objective function that is a fixed function of the parameters given a training set, often the log likelihood of training data under some parametric model. Viewed from the perspective of the generator G, the GAN training procedure does not involve a single, fixed objective function: G's objective is defined at any moment by the discriminator D, the

parameters of which are being continually adapted to both the data and to the current state of G. This can be considered a *learned objective function*, whereby the objective function for G is *automatically* adapted to the data distribution being estimated. The inductive bias for G is characterized by the family of functions from which D is chosen: G is optimized so as to elude detection via any statistical difference between p_g and p_{data} that D can learn to detect.

It is this property that is arguably responsible for the perceived visual quality of generated samples of GANs trained on natural images. Models trained via objective functions involving reconstruction terms, such as the variational autoencoder (Kingma and Welling, 2014; Rezende et al., 2014), implicitly commit to a static definition of sample plausibility. In the case of conditionally Gaussian likelihood, this takes the form of mean squared error, which is a particularly poor perceptual metric for natural image pixel intensities: it considers all perturbations of a given magnitude equivalent, without regard for the fact that changes in luminance which blur out sharp edges decrease the plausibility of the sample as a natural image much more than minor shifts in chroma across the entire image. While one popular approach in the case of models of natural images, and in many other domains, is to design the static objective so as to mitigate the mismatch between training criterion and the statistical properties of the domain, the solution offered by GANs is in some sense more universal: train D to detect and exploit any difference it can between the distributions of samples and real data, train G to outwit this new discriminator, and repeat. This often results in generated samples that more closely match human conceptions of saliency.

8.4 Extensions and refinements

Since the initial introduction of generative adversarial networks, the framework has been extended in several notable directions. Many of these rely on a straightforward extension to the *conditional* setting, where the generator and discriminator receive additional contextual inputs, first explored by Mirza and Osindero (2014). For example, in the aforementioned work, the authors train a class-conditional generator on the MNIST handwritten digits by feeding the network an additional input consisting of a "one-hot" vector indicating the desired class. The discriminator is

fed the generated or real image as well as the class label (the assigned label if the image is real, the desired label if the image is generated). Through training, the discriminator learns that in the presence of a given class label, the image should resemble instances of that class from the training data. Likewise, in order to succeed at fooling the discriminator, the generator must learn to use the class label input to inform the characteristics of its generated sample.

In pursuit of more realistic models of natural images, Denton et al. (2015) introduced a hierarchical model, dubbed LAPGAN, which interleaved conditional GAN generators with spatial upsampling in a Laplacian pyramid (Burt et al., 1983). The first generator, either class-conditional or traditional, is trained to generate a small thumbnail image. A fixed upsampling and blurring is performed and a second conditional generator, conditioned on the newly upsampled image, is trained to reproduce the *difference* between the image at the current resolution and the upsampled thumbnail. This process is iterated, with subsequent conditional generators predicting residuals at ever higher resolutions.

Also in the space of natural image generation, Radford et al. (2015) leveraged recent advances in the design and training of discriminative convolutional networks to successfully train a single adversarial pair to generated realistic images of relatively high resolution. These generator networks employ "fractionally strided convolutions", otherwise recognizable as the transpose operation of "valid"-mode strided convolution commonly used when backpropagating gradients through a strided convolutional layer, to learn their own upsampling operations. The authors identify a set of architectural constraints on the generator and discriminator which allow for relatively stable training, including the elimination of downsampling in favour of strided convolution in the discriminator the use of the bounded tanh function at the generator output layer, careful application of batch normalization (Ioffe and Szegedy, 2015) and the use of rectified linear units (Jarrett et al., 2009; Glorot et al., 2011) and leaky rectified linear units (Maas et al., 2013) throughout the generator and discriminator, respectively. Inspired by recent work on word embeddings (e.g. Mikolov et al. (2013)), the authors also interrogate the latent representations, i.e. samples from $p(\mathbf{z})$, and find that they obey surprising arithmetic properties when trained on a dataset of faces.

8.5 Hybrid models

A recent body of work has examined the combination of the adversarial network training criterion with other formalisms, notably autoencoders. Larsen et al. (2015) combine a GAN with a variational autoencoder (VAE) (Kingma and Welling, 2014; Rezende et al., 2014), dispensing with the VAE's reconstruction error term in favor of an squared error expressed in the space of the discriminator's hidden layers, combining the resulting modified VAE objective with the usual GAN objective. Makhzani et al. (2015) employs an adversarial cost as a regularizer on the hidden layer representation of a conventional autoencoder, forcing the aggregate posterior distribution of the hidden layer to match a particular synthetic distribution. This formulation closely resembles the VAE. The VAE maximizes a lower bound on the log-likelihood that includes both a reconstruction term and terms regularizing the variational posterior to resemble the model's prior distribution over the latent variables. The adversarial autoencoder removes the regularization term and uses the adversarial game to enforce the desired conditions.

The adversarial network paradigm has also been extended in the direction of supervised and semi-supervised learning. Springenberg (2016) generalizes the convention adversarial network setting to employ a categorical (softmax) output layer in the discriminator. The discriminator and generator compete to shape the *entropy* of this distribution while respecting constraints on its marginal distribution, and an optional likelihood term can add semantics to this output layer if class labels are available. Sutskever et al. (2015) propose an unsupervised criterion designed expressly with the intent of improving performance on downstream supervised tasks in settings where the space of possible outputs is large, and it is easy to obtain independent examples from both the input and output domains. The proposed supervised mapping is adversarially trained to have an output distribution resembling the distribution of independent output domain examples.

8.6 Beyond generative modeling

Generative adversarial networks were originally introduced in order to provide a means of performing generative modeling. The idea has since proven to be more

general. Adversarial pairs of networks may in fact be used for a broad range of tasks.

Two recent methods have shown that the adversarial framework can be used to impose desired properties on the features extracted by a neural network. The feature extractor can be thought of as analogous to the generator in the GAN framework. A second network, analogous to the discriminator, then tries to obtain some forbidden information from the extracted features. The feature extractor is then trained to learn features that are both useful for some original task, such as classification, and that yield little information to the second network. Ganin and Lempitsky (2015) use this approach for domain adaptation. The second network attempts to predict which domain the input was drawn from. When the feature extractor is trained to fool this network, it is forced to learn features that are invariant to the choice of input domain. Edwards and Storkey (2015) use a similar technique to learn representations that do not contain private information. In this case, the second network attempts to recover the private information from the representation. This approach could be used to remove prejudice from a decision making process. For example, if a machine learning model is used to make hiring decisions, it should not use protected information such as the race or gender of applicants. If the machine learning model is trained on the decisions made by human hiring managers, and if the previous hiring managers made biased decisions, the machine learning model could discover other features of the candidates that are correlated with their race or gender. By applying the method of Edwards and Storkey (2015), the machine learning model is encouraged to remove features that have a statistical relationship with the protected information, ideally leading to more fair decisions.

8.7 Discussion

The staggering gains in many application areas brought by the introduction of deep neural networks have inspired much excitement and widespread adoption. In addition to remarkable success tackling difficult supervised classification tasks, it is often the case that even misclassifications the errors made by state-of-the-art neural networks appear to be quite reasonable (as remarked, for example, by Krizhevsky

et al. (2012)). The existence of adversarial examples as a problem plaguing a wide variety of model families suggests surprising deficits both in the degree to which these models understand their tasks, and to which human practitioners truly understand their models. Research into such phenomena can yield immediate gains in robustness and resistance to attack for neural networks deployed in commercial and industrial systems, as well as guide research into new model classes which naturally resist such perturbation through a deeper comprehension of the learning task.

Simultaneously, the adversarial perspective can be fruitfully leveraged for tasks other than simple supervised learning. While the focus of generative modeling in the past has often been on models that directly optimize likelihood, many application domains express a need for realistic synthesis, including the generation of speech waveforms, image and video inpainting and super-resolution, the procedural generation of video game assets, and forward prediction in model-based reinforcement learning. Recent work (Theis et al., 2015) suggests that these goals may be at odds with this likelihood-centric paradigm. Generative adversarial networks and their extensions provide one avenue attack on these difficult synthesis problems with an intuitively appealing approach: to learn to generate convincingly, aim to fool a motivated adversary. An important avenue for future research concerns the quantitative evaluation of generative models intended for synthesis; particular desiderata include generic, widely applicable evaluation procedures which nonetheless can be made to respect domain-specific notions of similarity and verisimilitude.

9 Prologue to Fourth Article

9.1 Article Details

Improving generative adversarial networks with denoising feature matching. David Warde-Farley and Yoshua Bengio. *Proceedings of the 4th International Conference on Learning Representations (ICLR '17).*

Personal Contribution.

I conceived of the method, did all implementation and ran all experiments. I wrote the majority of the manuscript, with assistance from Yoshua Bengio.

9.2 Context

Generative adversarial networks had, by 2016, become arguably the most popular area of research in unsupervised machine learning, but their shortcomings, evident to us when first preparing the work in Goodfellow et al. (2014), seemed nearly as pronounced. Instabilities in training, in particular pathologies such as "mode collapse", were commonplace. The lack of access to an explicit density made objective, quantitative evaluation difficult and virtually absent from the literature. GANs were capable of synthesizing compelling images from relatively narrow domains such as photographs of human faces or bedroom scenes, but failed to reproduce "object-like" patterns when trained on more diverse collections.

The work in this chapter was directly precipitated by Salimans et al. (2016), which introduced a collection of heuristics which could synthesize compellingly when trained on diverse collections of natural images, but their success relied upon making use of class labels when training the discriminator, an unsatisfactory proposition from the perspective of unsupervised learning research. The same work introduced a heuristic, quantitative measure of sample quality known as the *Inception*

score, which correlates well with human judgements of sample quality and diversity, a welcome contribution in an area of research that was growing increasingly reliant on subjective, qualitative evaluation.[i]

9.3 Contributions

This work introduces a technique that augments the training criterion of generative adversarial networks with an additional training signal based on the reconstructions of a denoising auto-encoder trained in the feature embedding learned as a side effect of discriminator training. We demonstrate that the method qualitatively succeeds in reproducing object categories, and quantitatively rivals the method in Salimans et al. (2016) in terms of Inception score.

9.4 Recent Developments

As of this writing, denoising feature matching is still the method with the best reported Inception score on CIFAR10, although results in the literature have qualitatively improved a great deal.

Perhaps the most notable development since this article was published is the introduction of Wasserstein GANs (Arjovsky et al., 2017), which aim to alleviate certain theoretical shortcomings of the original formulation, observed in Arjovsky and Bottou (2017). The crude strategy in Arjovsky et al. (2017) of bounding the Lipschitz constant of the discriminator by clipping the absolute value of each weight was followed relatively soon by Gulrajani et al. (2017), who introduced a gradient penalty formulation towards the same end. Interestingly, another recently proposed method (Kodali et al., 2017) re-derives the gradient penalty, evaluated at a different set of points, from a game theoretic perspective, on top of the original GAN formulation. This suggests that gradient penalties may prove an important

i. The original GAN manuscript evaluated log likelihoods estimates using a method based on Parzen density estimation (Breuleux et al., 2011), but the community's consensus is that this method is unreliable in high-dimensional spaces; see Theis et al. (2015) and the quantitative analysis in Wu et al. (2017).

ingredient going forward, independent of the underlying adversarial game or other objective function in use.

Also notable is the work of Wu et al. (2017), who introduced a protocol based on annealed importance sampling (Neal, 2001) for evaluating lower bounds on the test log likelihood. While this technique is well known in the field of Bayesian statistics, they demonstrated a successful application to the family of "decoder-based" models, i.e. directed models where the conditional likelihood is parameterized by a neural network. They predictably showed that variational autoencoders (Kingma and Welling, 2014) outperform GANs (trained in the conventional fashion) in terms of test log likelihood, but also that the gap between the train and test log likelihood for the GANs they evaluated was relatively small, suggesting that overfitting (at least on simple datasets such as MNIST) was perhaps less of a problem than commonly believed.

10 Improving Generative Adversarial Networks with Denoising Feature Matching

10.1 Introduction

Generative adversarial networks (Goodfellow et al., 2014) (GANs) have become well known for their strength at realistic image synthesis. The objective function for the generative network is an implicit function of a learned discriminator network, estimated in parallel with the generator, which aims to tell apart real data from synthesized. Ideally, the discriminator learns to capture distinguishing features of real data, which the generator learns to imitate, and the process iterates until real data and synthesized data are indistinguishable.

In practice, GANs are well known for being quite challenging to train effectively. The relative model capacities of the generator and discriminator must be carefully balanced in order for the generator to effectively learn. Compounding the problem is the lack of an unambiguous and computable convergence criterion. Nevertheless, particularly when trained on image collections from relatively narrow domains such as bedroom scenes (Yu et al., 2015) and human faces (Liu et al., 2015), GANs have been shown to produce very compelling results.

For diverse image collections comprising a wider variety of the visual world, the results have generally been less impressive. For example, samples from models trained on ImageNet (Russakovsky et al., 2014) roughly match the local and global statistics of natural images but yield few recognizable objects. Recent work (Salimans et al., 2016) has sought to address this problem by training the discriminator in a semi-supervised fashion, granting the discriminator's internal representations knowledge of the class structure of (some fraction of) the training data it is presented. This technique markedly increases sample quality, but is unsatisfying from the perspective of GANs as a tool for unsupervised learning.

We propose to augment the generator's training criterion with a second training objective which guides the generator towards samples more like those in the

training set by explicitly modeling the data density in addition to the adversarial discriminator. Rather than deploy a second computationally expensive convolutional network for this task, the additional objective is computed in the space of features learned by the discriminator. In that space, we train a denoising auto-encoder, a family of models which is known to estimate the energy gradient of the data on which it is trained. We evaluate the denoising auto-encoder on samples drawn from the generator, and use the "denoised" features as targets – nearby feature configurations which are more likely than those of the generated sample, according to the distribution estimated by the denoiser.

We show that this yields generators which consistently produce recognizable objects on the CIFAR-10 dataset without the use of label information as in Salimans et al. (2016). The criterion appears to improve stability and possesses a degree of natural robustness to the well known "collapse" pathology. We further investigate the criterion's performance on two larger and more diverse collections of images, and validate our qualitative observations quantitatively with the *Inception score* proposed in Salimans et al. (2016).

10.2 Background

10.2.1 Generative adversarial networks

The generative adversarial networks paradigm (Goodfellow et al., 2014) estimates generative samplers by means of a training procedure which pits a *generator* G against a *discriminator* D. D is trained to tell apart training examples from samples produced by G, while G is trained to increase the probability of its samples being incorrectly classified as data. In the original formulation, the training procedure defines a continuous minimax game

$$\arg\min_{G} \quad \arg\max_{D} \mathbb{E}_{\mathbf{x}\sim\mathcal{D}} \log D(\mathbf{x}) + \mathbb{E}_{\mathbf{z}\sim p(\mathbf{z})} \log\left(1 - D\left(G(\mathbf{z})\right)\right) \tag{10.1}$$

where $\mathcal{D}$ is a data distribution on $\mathbb{R}^n$, D is a function that maps $\mathbb{R}^n$ to the unit interval, and G is a function that maps a noise vector $\mathbf{z} \in \mathbb{R}^m$, drawn from a simple

distribution $p(\mathbf{z})$, to the ambient space of the training data, $\mathbb{R}^n$. The idealized algorithm can be shown to converge and to minimize the Jensen-Shannon divergence between the data generating distribution and the distribution parameterized by G.

Goodfellow et al. (2014) found that in practice, minimizing (10.1) with respect to the parameters of G proved difficult, and elected instead to optimize an alternate objective,

$$\arg\max_G \mathbb{E}_{\mathbf{z}\sim p(\mathbf{z})} \log D\left(G(z)\right) \tag{10.2}$$

at the same time as D is optimized as above. $\log D(G(\mathbf{z}))$ yields more favourably scaled per-sample gradients for G when D confidently identifies a sample as counterfeit, avoiding the vanishing gradients arising in that case with the $-\log(1 - D(G(\mathbf{z})))$ objective.

Subsequent authors have investigated applications and extensions of GANs; for a review of this body of literature, see Warde-Farley and Goodfellow (2016). Of particular note for our purposes is Radford et al. (2015), who provide a set of general guidelines for the successful training of generative adversarial networks, and Salimans et al. (2016), who build upon these techniques with a number of useful heuristics and explore a variant in which the discriminator D is trained to correctly classify *labeled* training data, resulting in gradients with respect to the discriminator evidently containing a great deal of information relevant to generating "object-like" samples.

10.2.2 Challenges and Limitations of GANs

While Goodfellow et al. (2014) provides a theoretical basis for the GAN criterion, the theory relies on certain assumptions that are not satisfied in practice. Proofs demonstrate convergence of the GAN criterion in the unconstrained space of arbitrary functions; in practice, finitely parameterized families of functions such as neural networks are employed. As a consequence, the "inner loop" of the idealized algorithm – maximizing (10.1) with respect to (the parameters of) D, is infeasible to perform exactly, and in practice only one or a few gradient steps stand in for this maximization. This results in a *de facto* criterion for G which minimizes a lower bound on the correct objective (Goodfellow, 2014).

A commonly observed failure mode is that of full or partial collapse, where G

maps a large fraction of probable regions under $p(\mathbf{z})$ to only a few, low-volume regions of $\mathbb{R}^n$; in the case of images, this manifests as the appearance of many near-duplicate images in independent draws from G, as well as a lower diversity of samples and modes than what is observed in the dataset. As G and D are typically trained via mini-batch stochastic gradient descent, several authors have proposed heuristics that penalize such duplication within each mini-batch (Salimans et al., 2016; Zhao et al., 2016).

GANs represent a departure from traditional probabilistic models based on maximum likelihood and its approximations in that they parameterize a sampler directly and lack a closed form for the likelihood. This makes objective, quantitative evaluation difficult. While previous results in the literature have reported approximate likelihoods based on Parzen window estimates, Theis et al. (2015) has convincingly argued that these estimates can be quite misleading for high-dimensional data. In this work, we adopt the *Inception score* proposed by Salimans et al. (2016), which uses a reference Inception convolutional neural network (Szegedy et al., 2015) to compute

$$I(\{\mathbf{x}\}_1^N) = \exp\left(\mathbb{E}\left[D_{KL}(p(y|\mathbf{x})\|p(y)\right]\right)) \tag{10.3}$$

where $p(y|\mathbf{x})$ is provided by the output of the Inception network and $p(y) = \int_{\mathbf{x}} p(\mathbf{x})p(y|\mathbf{x})d\mathbf{x} \approxeq \frac{1}{N}\sum p(y|\mathbf{x}_i)$. Note that this score can be made larger by a low-entropy per-sample posterior (i.e. the Inception network classifies a given sample with greater certainty) as well as a higher entropy aggregate posterior (i.e. the Inception network identifies a wide variety of classes among the samples presented to it). Salimans et al. (2016) found this score correlated well with human evaluations of samplers trained on CIFAR-10; we therefore employ the Inception score here as a quantitative measure of visual fidelity of the samples, following the previous work's protocol of evaluating the average Inception score over 10 independent groups of 5,000 samples each. Error estimates correspond to standard deviations, in keeping with previously reported results.

10.3 Improving Unsupervised GAN Training On Diverse Datasets

In this work, we focus on the apparent difficulty of training GANs to produce "object-like" samples when trained on diverse collections of natural images. While Salimans et al. (2016) make progress on this problem by employing labeled data and training the discriminator, here we aim to make progress on the unsupervised case. Nevertheless, our methods would be readily applicable to supervised, semi-supervised or (with slight modifications) conditional setting.

We begin from the slightly subtle observation that in realistic manifestations of the GAN training procedure, the discriminator's (negative) gradient with respect to a sample points in a direction of (infinitesimal) local improvement with respect to the discriminator's estimate of the sample being data; it *does not* necessarily point in the direction of a draw from the data distribution. Indeed, the literature is replete with instances of gradient descent with respect to the input of a classification model, particularly wide-domain natural image classifiers, producing ghostly approximations to a particular class exemplar (Le et al., 2012; Erhan et al., 2009; Yosinski et al., 2015) when this procedure is carried out without additional guidance, to say nothing of the problems posed by adversarial examples (Szegedy et al., 2014; Goodfellow et al., 2014) and fooling examples (Nguyen et al., 2015).

While the gradient of the loss function defined by the discriminator may be a source of information mostly relevant to very local improvements, the discriminator itself is a potentially valuable source of compact descriptors of the training data. Many authors have noted the remarkable versatility of high-level features learned by convolutional networks (Donahue et al., 2014; Yosinski et al., 2014) and the degree to which high-level semantics can be reconstructed from even the deepest layers of a network (Dosovitskiy and Brox, 2016). Although non-stationary, the distribution of the high-level activations of the discriminator when evaluated on data is ripe for exploitation as an additional source of knowledge about salient aspects of the data distribution.

We propose in this work to track this distribution with a denoising auto-encoder $r(\cdot)$ trained on the discriminator's hidden states $\mathbf{h}$, when evaluated on training data. Alain and Bengio (2014) showed that a denoising auto-encoder trained on data from a distribution $q(\mathbf{h})$ estimates via $r(\mathbf{h}) - \mathbf{h}$ the gradient of the true log-

density, $\frac{\partial \log q(\mathbf{h})}{\partial \mathbf{h}}$. Hence, if we train the denoising auto-encoder on the transformed training data $\mathbf{h} = \Phi(\mathbf{x})$ with $\mathbf{x} \sim \mathcal{D}$, then $r(\Phi(\mathbf{x}')) - \Phi(\mathbf{x}')$ with $\mathbf{x}' = G(\mathbf{z})$ indicates in which direction $\mathbf{x}'$ should be changed in order to make $\mathbf{h} = \Phi(\mathbf{x}')$ more like those features seen with the data. Minimizing $||r(\Phi(\mathbf{x}')) - \Phi(\mathbf{x}')||^2$ with respect to $\mathbf{x}'$ would thus push $\mathbf{x}'$ towards higher probability configurations according to the data distribution in the feature space $\Phi(\mathbf{x})$. We thus evaluate the discriminator features $\Phi(\mathbf{x})$, and the denoising auto-encoder, on samples from the generator, and treat the denoiser's output reconstruction as a fixed target for the generator. We refer to this procedure as *denoising feature matching*, and employ it as a learning signal for the generator in addition to the traditional GAN generator objective.

Formally, let G be the generator parameterized by θ_G, and $D = d \circ \Phi$ be our discriminator composing feature extractor $\Phi(\cdot) : \mathbb{R}^n \to \mathbb{R}^k$ and a classifier $d(\cdot) : \mathbb{R}^k \to [0, 1]$. Let $C(\cdot) : \mathbb{R}^k \to \mathbb{R}^k$ be a corruption function to be applied at the input of the denoising auto-encoder when it is trained to denoise. The parameters of the discriminator D, comprising the parameters of both d and Φ, is trained as in Goodfellow et al. (2014), while the generator is trained according to

$$\arg\min_{\theta_G} \mathbb{E}_{\mathbf{z}\sim p(\mathbf{z})} \left[\lambda_{\text{denoise}} \|\Phi(G(\mathbf{z})) - r(\Phi(G(\mathbf{z})))\|^2 - \lambda_{\text{adv}} \log D(G(z)) \right] \tag{10.4}$$

where $r(G(\mathbf{z}))$ is treated as constant with respect to gradient computations. Simultaneously, the denoiser $r(\cdot)$ is trained according to the objective

$$\arg\min_{\theta_r} \mathbb{E}_{\mathbf{x}\sim\mathcal{D}} \|\Phi(\mathbf{x}) - r(C(\Phi(\mathbf{x})))\|^2 \tag{10.5}$$

10.3.1 Effect of Φ

The theory surrounding denoising auto-encoders applies when estimating a denoising function from a data distribution $p(\mathbf{x})$. Here, we propose to estimate the denoising auto-encoder in the space of discriminator features, giving rise to a distribution $q(\Phi(\mathbf{x}))$. A natural question is what effect this has on the gradient being backpropagated. This is difficult to analyze in general, as for most choices the mapping Φ will not be invertible, though it is instructive to examine the invertible case. Assuming an invertible $\Phi : \mathbb{R}^n \to \mathbb{R}^n$, let $J = \frac{\partial \Phi(\mathbf{x})}{\partial \mathbf{x}}$ be the Jacobian of Φ, and $q(\Phi(\mathbf{x})) = p(\mathbf{x})|J|$. By the inverse function theorem, J is also invertible (and is in fact the Jacobian of the inverse Φ^{-1}). Applying the chain rule and re-arranging

terms, taking advantage of the invertibility of J, we arrive at a straightforward relationship between the score of q and the score of p:

$$\frac{\partial \log q(\Phi(\mathbf{x}))}{\partial \Phi(\mathbf{x})} = \frac{\partial \log \left[p(\mathbf{x}) \left|J\right|\right]}{\partial \Phi(\mathbf{x})} \tag{10.6}$$

$$= \frac{\partial \log p(\mathbf{x})}{\partial \Phi(\mathbf{x})} + \frac{\partial \log \left|\frac{\partial \Phi(\mathbf{x})}{\partial \mathbf{x}}\right|}{\partial \Phi(\mathbf{x})} \tag{10.7}$$

$$= \left(\frac{\partial \log p(\mathbf{x})}{\partial \mathbf{x}} + \frac{\partial \log \left|J\right|}{\partial \mathbf{x}}\right) J^{-1} \tag{10.8}$$

where

$$\frac{\partial \log \left|J\right|}{\partial x_k} = \operatorname{Tr}\left(J^{-1} \frac{dJ}{dx_k}\right) \tag{10.9}$$

and $\frac{dJ}{dx_k}$ is a matrix of scalar derivatives of elements of J with respect to x_k. Thus, we see that the gradient backpropagated to the generator in an ideal setting is the gradient of the data distribution $p(\mathbf{x})$ along with an additive term which accounts for the changes in the rate of volume expansion/contraction in Φ locally around $\mathbf{x}$. In practice, Φ is not invertible, but the added benefit of the denoiser-targeted gradient appears to reduce underfitting to the modes of p in the generator, irrespective of any distortions Φ may introduce.

10.4 Related work

Denoising feature matching was originally inspired by *feature matching* introduced by Salimans et al. (2016) as an alternative training criterion for GAN generators, namely (in our notation)

$$\arg\min_{\theta_G} \left| \left\| \mathbb{E}_{\mathbf{x}\sim\mathcal{D}} \left[\Phi(\mathbf{x})\right] - \mathbb{E}_{\mathbf{z}\sim p(z)} \left[\Phi(G(\mathbf{z}))\right] \right\| \right|^2 \tag{10.10}$$

Feature matching is equivalent to linear maximum mean discrepancy (Gretton et al., 2006), employing linear first moment matching in the space of discriminator features $\Phi(\cdot)$ rather than the more familiar kernelized formulation. When

performed on features in the penultimate layer, Salimans et al. (2016) found that the feature matching criterion was useful for the purpose of improving results on semi-supervised classification, using classification of samples from the generator as a sophisticated form of data augmentation. Feature matching was, however, less successful at producing samples with high visual fidelity. This is somewhat unsurprising given that the criterion is insensitive to higher-order statistics of the respective feature distributions. Indeed, a degenerate G which deterministically reproduces a single sample $\hat{\mathbf{m}}$ such that $\Phi(\hat{\mathbf{m}}) = \mathbb{E}_{\mathbf{x}\in\mathcal{D}}\Phi(\mathbf{x})$ trivially minimizes (10.10); in practice the joint training dynamics of D and G do not appear to yield such degenerate solutions.

Rather than aiming to merely reduce linear separability between data and samples in the feature space defined by $\Phi(\cdot)$, denoising feature matching selects a more probable (according to the feature distribution implied by the data, as captured by the denoiser) feature space target for each sample produced by G and regresses G towards it. While an early loss of entropy in G could result in the generator locking on to one or a few attractors in the denoiser's energy landscape, we observe that this does not happen when used in conjunction with the traditional GAN objective, and in fact that the combination of the two objectives is notably robust to the collapses often observed in GAN training, even without taking additional measures to prevent them.

This work also draws inspiration from Alain and Bengio (2014), which showed that a suitably trained denoiser learns an operator which locally maps a sample towards regions of high probability under the data distribution. They further showed that a suitably trained [i] reconstruction function $r(\cdot)$ behaves such that

$$r(\mathbf{x}) - \mathbf{x} \propto \frac{\partial \log p(\mathbf{x})}{\partial \mathbf{x}} \tag{10.11}$$

That is, $r(\mathbf{x}) - \mathbf{x}$ estimates the score of the data generating distribution, up to a multiplicative constant. Our use of denoising auto-encoders necessarily departs from idealized conditions in that the denoiser is estimated online from an ever-changing distribution of features.

Several approaches to GAN-like models have cast the problem in terms of learn-

i. In the limit of infinite training data, with isotropic Gaussian noise of some standard deviation σ.

ing an energy function. Kim and Bengio (2016) extends GANs by modeling the data distribution simultaneously with an energy function parameterized by a deep neural network (playing the role of the discriminator) and the traditional generator, carrying out learning with a learning rule resembling that of the Boltzmann machine (Ackley et al., 1985), where the "negative phase" gradient is estimated from samples from the generator. The energy-based GAN formulation of Zhao et al. (2016) resembles our work in their use of an auto-encoder which is trained to faithfully reconstruct (in our case, a corrupted, function of) the training data. The energy-based GAN *replaces* the discriminator with an auto-encoder, which is trained to assign low energy (L_2 reconstruction error) to training data and higher energy to samples from G. To discourage generator collapses, a "pull-away term" penalizes the normalized dot product in a feature space defined by the auto-encoder's internal representation. In this work, we preserve the discriminator, trained in the usual discriminative fashion, and in fact preserve the traditional generator loss, instead augmenting it with a source of complementary information provided by targets obtained from the denoiser. The energy-based GAN can be viewed as training the generator to seek fixed points of the autoencoding function (i.e. by backpropagating through the decoder and encoder in order to decrease reconstruction error), whereas we treat the output of $r(\cdot)$ as constant with respect to the optimization as in Lee et al. (2015). That is to say, rather than using backpropagation to steer the dynamics of the autoencoder, we instead employ our denoising autoencoder to augment the gradient information obtained by ordinary backpropagation.

Closest to our own approach, concurrent work on model-based super-resolution by Sønderby et al. (2016) trains a denoising auto-encoder on high-resolution ground truth and evaluates it on synthesized super-resolution images, using the difference between the original synthesized image and the denoiser's output as an additional training signal for refining the output of the super-resolution network. Both Sønderby et al. (2016) and our own work are motivated by the results of Alain and Bengio (2014) discussed above. Aside from addressing a different application area, our denoiser is learned on-the-fly from a high-level feature representation which is itself learned.

10.5 Experiments

We evaluate denoising feature matching on learning synthesis models from three datasets of increasing diversity and size: CIFAR-10, STL-10, and ImageNet. Although several authors have described GAN-based image synthesis models operating at 128×128 (Salimans et al., 2016; Zhao et al., 2016) and 256×256 (Zhao et al., 2016) resolution, we carry out our investigations at relatively low resolutions, both for computational ease and because we believe that the problem of unconditional modeling of diverse image collections is not well solved even at low resolutions; making progress in this regime is likely to yield insights that apply to the higher-resolution case.

In all experiments, we employ isotropic Gaussian corruption noise with $\sigma = 1$. Although we experimented with annealing σ towards 0 (as also performed in Sønderby et al. (2016)), an annealing schedule which consistently outperformed fixed noise remained elusive. We experimented with convolutional denoisers, but our best results to date were obtained with deep, fully-connected denoisers using the ReLU nonlinearity on the penultimate layer of the discriminator. The number of hidden units was fixed to the same value in all denoiser layers, and the procedure is apparently robust to this hyperparameter choice, as long as it is greater than or equal to the input dimensionality.

Our generator and discriminator architectures follow the methods outlined in Radford et al. (2015). Accordingly, batch normalization (Ioffe and Szegedy, 2015) was used in the generator and discriminator in the same manner as Radford et al. (2015), and in all layers of the denoiser except the output layer. In particular, as in Radford et al. (2015), we separately batch normalize data and generator samples for the discriminator and denoiser with respect to each source's statistics. We calculate updates with respect to all losses with the parameters of all three networks fixed, and update all parameters simultaneously.

All networks were trained with the Adam optimizer (Kingma and Ba, 2014) with a learning rate of 10^{-4} and $\beta_1 = 0.5$. The Adam optimizer is scale invariant, and so it suffices to e.g. tune λ_{denoise} and fix λ_{adv} to 1. In our experiments, we set $\lambda_{denoise}$ to $0.03/n_h$, where n_h is the number of discriminator hidden units fed as input to the denoiser; this division decouples the scale of the first term of (10.4) from the dimensionality of the representation used, reducing the need to adjust this

hyperparameter simply because we altered the architecture of the discriminator.

10.5.1 CIFAR-10

CIFAR-10 (Krizhevsky and Hinton, 2009) is a small, well-studied dataset consisting of 50,000 32×32 pixel RGB training images and 10,000 test images from 10 classes: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck.

Samples from our model trained on CIFAR-10 are shown in Figure 10.1, and Inception scores for several methods, including those reported in Salimans et al. (2016) and scores computed from samples generated from a model presented in Dumoulin et al. (2016), are presented in Table 10.1. We achieve a mean Inception score of 7.72, falling slightly short of Salimans et al. (2016), which employed a supervised discriminator network (the same work reports a score of $4.36 \pm .04$ when labels are omitted from their training procedure). Qualitatively, the samples include recognizable cars, boats and various animals. The best performing generator network consisted of the 32×32 ImageNet architecture from Radford et al. (2015) with half the number of parameters at each layer, and less than 40% of the parameters of the CIFAR-10 generator presented in Salimans et al. (2016).

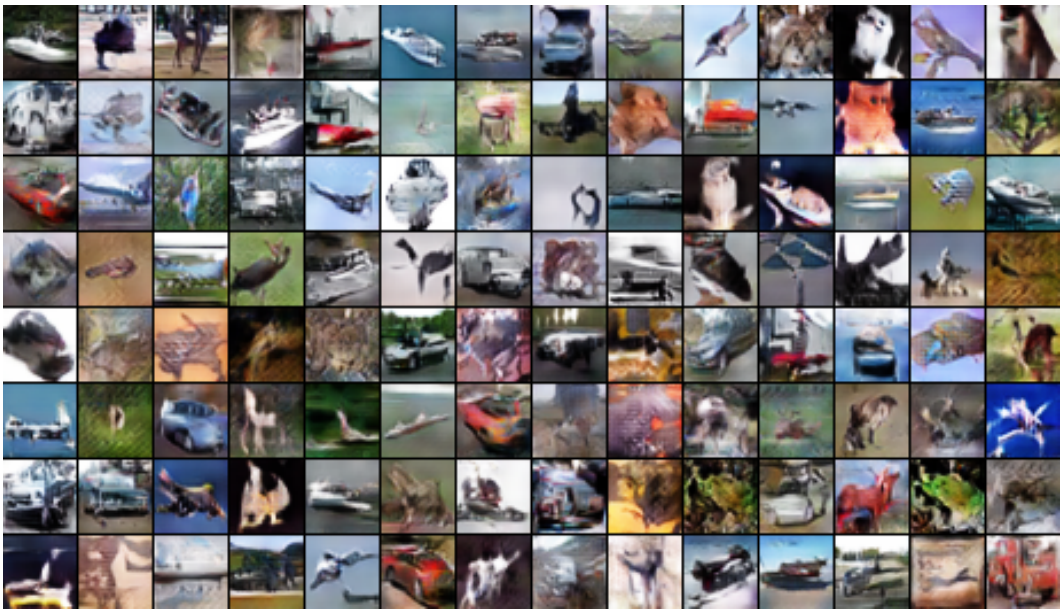

Figure 10.1 – Samples generated from a model trained with denoising feature matching on CIFAR10.

Real data*	Semi-supervised	Unsupervised	
	Improved GAN (Salimans *et al*)*	ALI (Dumoulin *et al*)†	Ours
11.24 ± .12	8.09 ± .07	5.34 ± 0.05	7.72 ± 0.13

Table 10.1 – Inception scores for models of CIFAR-10. * as reported in Salimans et al. (2016); semi-supervised † computed from samples drawn using author-provided model parameters and implementation.

10.5.2 STL-10

STL-10 (Coates et al., 2011) is a dataset consisting of a small labeled set and larger (100,000) unlabeled set of 96×96 RGB images. The unlabeled set is a subset of ImageNet that is more diverse than CIFAR-10 (or the labeled set of STL-10), but less diverse than full ImageNet. We downsample by a factor of 2 on each dimension and train our networks at 48×48. Inception scores for our model and a baseline, consisting of the same architecture trained without denoising feature matching (both trained for 50 epochs), are shown in Table 10.2. Samples are displayed in Figure 10.2.

Real data	Ours	GAN Baseline
26.08 ± .26	8.51 ± 0.13	7.84 ± .07

Table 10.2 – Inception scores for models of the unlabeled set of STL-10.

10.5.3 ImageNet

The ImageNet database (Russakovsky et al., 2014) is a large-scale database of natural images. We train on the designated training set of the most widely used release, the 2012 ImageNet Large Scale Visual Recognition Challenge (ILSVRC2012), consisting of a highly unbalanced split among 1,000 object classes. We preprocess the dataset as rescaled central crops following the procedure of Krizhevsky et al. (2012), except at 32×32 resolution to facilitate comparison with Radford et al. (2015).

ImageNet poses a particular challenge for unsupervised GANs due to its high level of diversity and class skew. With a generator and discriminator architecture identical to that used for the same dataset in Radford et al. (2015), we achieve

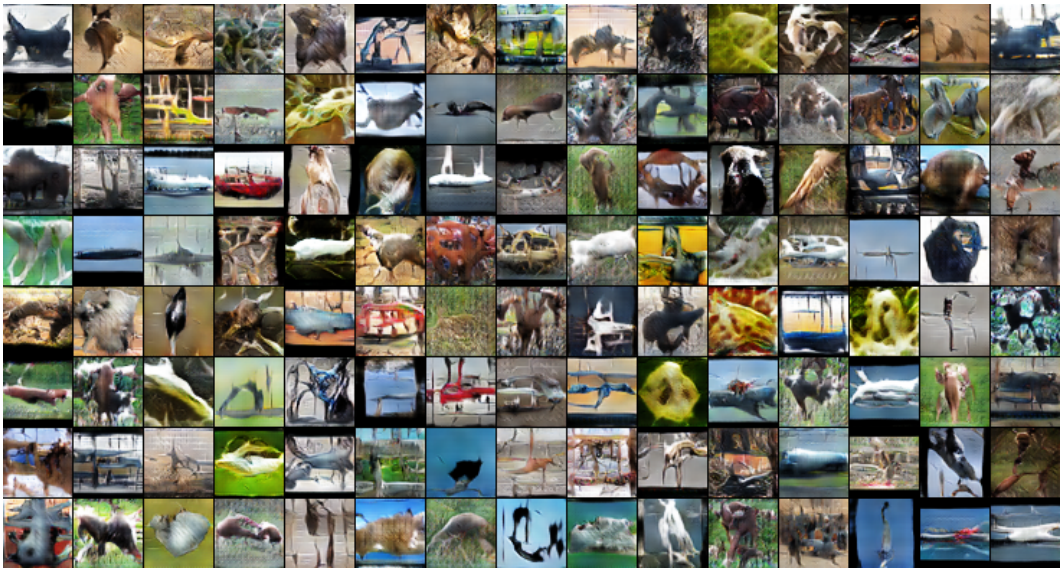

Figure 10.2 – Samples from a model trained with denoising feature matching on the unlabeled portion of the STL-10 dataset.

a higher Inception score using denoising feature matching, using denoiser with 10 hidden layers of 2,048 rectified linear units each. Both fall far short of the score assigned to real data at this resolution; there is still plenty of room for improvement. Samples are displayed in Figure 10.3.

Real data	Radford *et al**	Ours
25.78 ± .47	8.83 ± 0.14	9.18 ± .13

Table 10.3 – Inception scores for models of ILSVRC 2012 at 32×32 resolution. * computed from samples drawn using author-provided model parameters and implementation.

10.6 Discussion and Future Directions

We have shown that training a denoising model on high-level discriminator activations in a GAN, and using the denoiser to propose high-level feature targets for the generator, can usefully improve GAN image models. Higher Inception scores, as well as visual inspection, suggest that the procedure captures class-specific features of the training data in a manner superior to the original adversarial objective

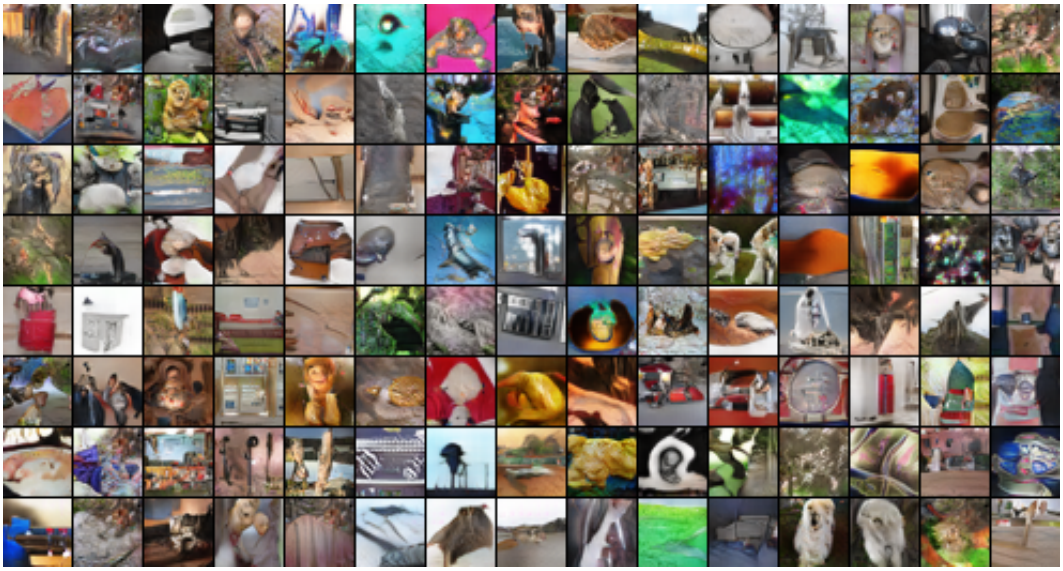

Figure 10.3 – Samples from our model of ILSVRC2012 at 32×32 resolution.

alone. That being said, we do not believe we are yet making optimal use of the paradigm. The non-stationarity of the feature distribution on which the denoiser is trained could be limiting the ability of the denoiser to obtain a good fit, and the information backpropagated to the generator is always slightly stale. Steps to reduce this non-stationarity may be fruitful; we experimented briefly with historical averaging as explored in Salimans et al. (2016) but did not observe a clear benefit thus far. Structured denoisers, including denoisers that learn an energy function for multiple hidden layers at once, could conceivably aid in obtaining a better fit. Learning a partially stochastic transition operator rather than a deterministic denoiser could conceivably capture interesting multimodalities that are "blurred" by a unimodal denoising function.

Our method is orthogonal and could conceivably be used in combination with several other GAN extensions. For example, methods incorporating an encoder component (Donahue et al., 2016; Dumoulin et al., 2016), various existing conditional architectures (Mirza and Osindero, 2014; Denton et al., 2015; Reed et al., 2016), or the semi-supervised variant employed in Salimans et al. (2016), could all be trained with an additional denoising feature matching objective.

We have proposed a useful heuristic, but a better theoretical grounding regarding how GANs are trained in practice is a necessary direction for future work,

including grounded criteria for assessing mode coverage and mass misassignment, and principled criteria for assessing convergence or performing early stopping.

Acknowledgments

We thank Ian Goodfellow, Laurent Dinh, Yaroslav Ganin and Kyle Kastner for helpful discussions. We thank Vincent Dumoulin and Ishmael Belghazi for making available code and model parameters used in comparison to ALI, as well as Alec Radford for making available the code and model parameters for his ImageNet model. We would like to thank Antonia Creswell and Hiroyuki Yamazaki for pointing out an error in the initial version of this manuscript, and anonymous reviewers for valuable feedback. We thank the University of Montreal and Compute Canada for the computational resources used for this investigation, as well as the authors of Theano (Al-Rfou et al., 2016), Blocks and Fuel (van Merriënboer et al., 2015). We thank CIFAR, NSERC, Google, Samsung and Canada Research Chairs for funding.

11 Discussion

This thesis has touched on many aspects of the training and deployment of feed-forward neural networks. We introduced maxout, a generalization of previous piecewise-linear activations and the first example of a per-unit "learned activation function". We experimentally investigated the popular dropout procedure for regularization and shed light on certain questions surrounding its efficacy. We proposed an effective strategy for augmenting a trained neural network classifier to improve its accuracy by adding capacity specifically designed to alleviate errors between frequently confused classes. Finally, we devised a hybrid training criterion for generative adversarial networks which improves their ability, measured qualitatively and quantitatively via the Inception score, to match the target distribution when trained on diverse collections of images. It is notable that each of the contributions addresses topics related to either overfitting or underfitting; issues surrounding optimization of these networks, and in particular certain phenomena related to generalization (Zhang et al., 2016), remain poorly understood in general, especially with regard to the nascent body of literature surrounding adversarial networks.

Maxout has made a lasting mark on the deep learning literature, inspiring many efforts at adaptive activation function design (Agostinelli et al., 2014; Clevert et al., 2015; Jin et al., 2016), though ReLU and its variants persist as the most popular choice among practitioners. Maxout remains a useful tool for augmenting the capacity of one layer of a neural network without altering the representation size fed to subsequent layers. In practice, little benefit has been observed for pool sizes greater than 2 for convolutional layers, and pool sizes greater than 5 for fully connected layers. More sophisticated strategies for combatting under-utilization of this capacity, including replacing or augmenting the (extremely simple) competition mechanism employed by maxout, may yet yield further improvements.

Dropout regularization continues to play an important role in applications of neural networks. While dropout proved an important ingredient in the work of Krizhevsky et al. (2012), which established convolutional neural networks as the

tool of choice in computer vision, it has fallen out of use in recent years, as tools such as batch normalization (Ioffe and Szegedy, 2015) and improved architectures such as residual networks (He et al., 2016) have improved generalization performance with a decreased need for explicit regularization.

The networks discussed in chapter 7 represented one step in the direction of task-adapted network architectures. While achieving their goal of improving fine-grained classification of a highly specialized architecture at minimal additional overhead, they did not leverage any form of conditional computation (Bengio et al., 2013). Previous attempts at large-scale gated architectures had suffered from a "cold start" problem, making it difficult to make good decisions about which modules to apply while simultaneously encouraging both an equitable distribution of responsibility and pathway-specific specialization. Very recently, large, "hard" mixtures of experts (Jacobs et al., 1991) have been successfully trained, both for large scale image classifiers (Gross et al., 2017) and as components in large, convolutional models for text processing (Shazeer et al., 2017). Principles for effectively training large conditional computation models are an important research direction for scaling deep learning towards general-purpose artificially intelligent agents.

Adversarial networks continue to be a growing area of research, but many fundamental questions remain unanswered. The single greatest challenge, in the opinion of this author, is tractable, broadly applicable evaluation methods for implicit generative models. The Inception score used in chapter 10, the only quantitative measure on which published results existed at the time, is unfortunately ill suited to detecting many kinds of pathologies: the entropy of the categorical distribution over classes is a poor proxy for assessing image quality (or resemblance to the class predicted by the maximal output), as it is well known that deep neural network classifier can be made to respond confidently to images that are well out of domain with respect to their training (Nguyen et al., 2015) and that even where realistic images are concerned, the predicted class label can be influenced by human-imperceptible perturbations (Szegedy et al., 2014; Goodfellow et al., 2014).

Importantly, the Inception score makes no direct comparison with the statistics of held-out test data (or indeed, any real data at all, except very indirectly via the classifier's training data). The Fréchet Inception distance proposed in Heusel et al. (2017) improves upon this somewhat by measuring the Fréchet between high-level features of the Inception network extracted from real data and samples, but it is

unclear to what extent applying this measure to held out data could reliably detect overfitting. To date, the only proposed method capable of detecting misallocation of density or overfitting to the training set is the AIS procedure outlined in Wu et al. (2017); in practice, this procedure requires considerable computational resources, making it impractical for large test sets and high-dimensional generated distributions. Methods which ease this evaluative burden, perhaps via principled combination of adversarial learning with other inductive principles (Rosca et al., 2017; Hu et al., 2017), could contribute greatly to the increased adoption and commodification of GAN-based methods.

Bibliography

Abdi, H. (2007). *Bonferroni and Sidak corrections for multiple comparisons*. Sage.

Ackley, D. H., G. E. Hinton, and T. J. Sejnowski (1985). A learning algorithm for boltzmann machines. *Cognitive science 9*(1), 147–169.

Agostinelli, F., M. Hoffman, P. Sadowski, and P. Baldi (2014). Learning activation functions to improve deep neural networks. *arXiv preprint arXiv:1412.6830*.

Aizerman, M. A., E. M. Braverman, and L. I. Rozonoer (1964). Theoretical foundations of the potential function method in pattern recognition learning. *Automation and Remote Control 25*, 821–837.

Al-Rfou, R., G. Alain, A. Almahairi, and et al. (2016). Theano: A python framework for fast computation of mathematical expressions. *CoRR abs/1605.02688*.

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

Arjovsky, M. and L. Bottou (2017). Towards principled methods for training generative adversarial networks. *arXiv preprint arXiv:1701.04862*.

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

Baldi, P. and K. Hornik (1989). Neural networks and principal component analysis: Learning from examples without local minima. *2*, 53–58.

Baldi, P. and P. J. Sadowski (2013). Understanding dropout. In *Advances in Neural Information Processing Systems 26*, pp. 2814–2822.

Bastien, F., P. Lamblin, R. Pascanu, J. Bergstra, I. J. Goodfellow, A. Bergeron, N. Bouchard, and Y. Bengio (2012). Theano: new features and speed improvements. Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop.

Bengio, Y. (2009). Learning deep architectures for AI. *2*(1), 1–127.

Bengio, Y. (2013). Deep learning of representations: Looking forward. In *Statistical Language and Speech Processing*, pp. 1–37. Springer.

Bengio, Y. and Y. LeCun (2007). Scaling learning algorithms towards AI. In L. Bottou, O. Chapelle, D. DeCoste, and J. Weston (Eds.), *Large Scale Kernel Machines*. MIT Press.

Bengio, Y., N. Léonard, and A. Courville (2013, August). Estimating or propagating gradients through stochastic neurons for conditional computation. *ArXiv e-prints abs/1308.3432*.

Bengio, Y., L. Yao, G. Alain, and P. Vincent (2013). Generalized denoising auto-encoders as generative models. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger (Eds.), *Advances in Neural Information Processing Systems 26*, pp. 899–907. Curran Associates, Inc.

Bergstra, J. and Y. Bengio (2012). Random search for hyper-parameter optimization. *J. Machine Learning Res. 13*, 281–305.

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

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

Bollacker, K., C. Evans, P. Paritosh, T. Sturge, and J. Taylor (2008). Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pp. 1247–1250. ACM.

Boser, B. E., I. M. Guyon, and V. N. Vapnik (1992). A training algorithm for optimal margin classifiers. In *COLT '92: Proceedings of the fifth annual workshop on Computational learning theory*, New York, NY, USA, pp. 144–152. ACM.

Bourlard, H. and Y. Kamp (1988). Auto-association by multilayer perceptrons and singular value decomposition. *59*, 291–294.

Breiman, L. (1994). Bagging predictors. *Machine Learning 24*(2), 123–140.

Breuleux, O., Y. Bengio, and P. Vincent (2011, August). Quickly generating representative samples from an RBM-derived process. *Neural Computation 23*(8), 2053–2073.

Buciluǎ, C., R. Caruana, and A. Niculescu-Mizil (2006). Model compression. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 535–541. ACM.

Burt, P. J., Edward, and E. H. Adelson (1983). The laplacian pyramid as a compact image code. *IEEE Transactions on Communications 31*, 532–540.

Cartwright, D. I. and M. J. Field (1978). A refinement of the arithmetic mean-geometric mean inequality. *Proceedings of the American Mathematical Society 71*(1), pp. 36–38.

Chung, F. R. (1997). *Spectral graph theory*, Volume 92. American Mathematical Soc.

Ciresan, D. C., U. Meier, L. M. Gambardella, and J. Schmidhuber (2010). Deep big simple neural nets for handwritten digit recognition. *Neural Computation 22*, 1–14.

Clevert, D.-A., T. Unterthiner, and S. Hochreiter (2015). Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*.

Coates, A., H. Lee, and A. Y. Ng (2011). An analysis of single-layer networks in unsupervised feature learning. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (AISTATS 2011)*.

Cortes, C. and V. Vapnik (1995). Support vector networks. *Machine Learning 20*, 273–297.

Dean, J., G. Corrado, R. Monga, K. Chen, M. Devin, Q. Le, M. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang, and A. Y. Ng (2012). Large scale distributed deep networks. In *NIPS'2012*.

Dempster, A. P., N. M. Laird, and D. B. Rubin (1977). Maximum-likelihood from incomplete data via the EM algorithm. *Journal of Royal Statistical Society B 39*, 1–38.

Denton, E. L., S. Chintala, R. Fergus, et al. (2015). Deep generative image models using a laplacian pyramid of adversarial networks. In *Advances in neural information processing systems*, pp. 1486–1494.

Dinh, L., J. Sohl-Dickstein, and S. Bengio (2016). Density estimation using real nvp. *arXiv preprint arXiv:1605.08803*.

Donahue, J., Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell (2014). Decaf: A deep convolutional activation feature for generic visual recognition. In *ICML*, pp. 647–655.

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

Dosovitskiy, A. and T. Brox (2016, June). Inverting visual representations with convolutional networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

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

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

Edwards, H. and A. J. Storkey (2015). Censoring representations with an adversary. *CoRR abs/1511.05897*.

Erhan, D., Y. Bengio, A. Courville, and P. Vincent (2009). Visualizing higher-layer features of a deep network. *University of Montreal 1341*.

Ganin, Y. and V. Lempitsky (2015). Unsupervised domain adaptation by backpropagation. In *ICML'2015*.

Germain, M., K. Gregor, I. Murray, and H. Larochelle (2015). Made: Masked autoencoder for distribution estimation. *arXiv preprint arXiv:1502.03509*.

Glorot, X. and Y. Bengio (2010, May). Understanding the difficulty of training deep feedforward neural networks. In *JMLR W&CP: Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (AISTATS 2010)*, Volume 9, pp. 249–256.

Glorot, X., A. Bordes, and Y. Bengio (2011, April). Deep sparse rectifier neural networks. In *JMLR W&CP: Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (AISTATS 2011)*.

Goodfellow, I., J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio (2014). Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger (Eds.), *Advances in Neural Information Processing Systems 27*, pp. 2672–2680. Curran Associates, Inc.

Goodfellow, I. J. (2014). On distinguishability criteria for estimating generative models. In *International Conference on Learning Representations, Workshops Track*.

Goodfellow, I. J., Y. Bulatov, J. Ibarz, S. Arnoud, and V. Shet (2014). Multi-digit number recognition from Street View imagery using deep convolutional neural networks. In *International Conference on Learning Representations*.

Goodfellow, I. J., M. Mirza, A. Courville, and Y. Bengio (2013, December). Multi-prediction deep Boltzmann machines. In *Advances in Neural Information Processing Systems 26 (NIPS'13)*. NIPS Foundation (http://books.nips.cc).

Goodfellow, I. J., J. Shlens, and C. Szegedy (2014). Explaining and harnessing adversarial examples. *CoRR abs/1412.6572*.

Goodfellow, I. J., D. Warde-Farley, P. Lamblin, V. Dumoulin, M. Mirza, R. Pascanu, J. Bergstra, F. Bastien, and Y. Bengio (2013). Pylearn2: a machine learning research library. *arXiv preprint arXiv:1308.4214*.

Goodfellow, I. J., D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio (2013). Maxout networks. In S. Dasgupta and D. McAllester (Eds.), *Proceedings of the 30th International Conference on Machine Learning (ICML'13)*, pp. 1319–1327. ACM.

Gregor, K., I. Danihelka, A. Mnih, C. Blundell, and D. Wierstra (2014). Deep autoregressive networks. In *International Conference on Machine Learning (ICML'2014)*.

Gretton, A., K. M. Borgwardt, M. Rasch, B. Schölkopf, and A. J. Smola (2006). A kernel method for the two-sample-problem. In *Advances in neural information processing systems*, pp. 513–520.

Gross, S., M. Ranzato, and A. Szlam (2017). Hard mixtures of experts for large scale weakly supervised vision. *arXiv preprint arXiv:1704.06363*.

Gulrajani, I., F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville (2017). Improved training of wasserstein gans. *arXiv preprint arXiv:1704.00028*.

Gutmann, M. and A. Hyvarinen (2010). Noise-contrastive estimation: A new estimation principle for unnormalized statistical models.

Hahnloser, R. H. R. (1998). On the piecewise analysis of networks of linear threshold neurons. *Neural Networks 11*(4), 691–697.

Havaei, M., A. Davy, D. Warde-Farley, A. Biard, A. Courville, Y. Bengio, C. Pal, P.-M. Jodoin, and H. Larochelle (2017). Brain tumor segmentation with deep neural networks. *Medical image analysis 35*, 18–31.

He, K., X. Zhang, S. Ren, and J. Sun (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778.

Heusel, M., H. Ramsauer, T. Unterthiner, B. Nessler, G. Klambauer, and S. Hochreiter (2017). Gans trained by a two time-scale update rule converge to a nash equilibrium. *arXiv preprint arXiv:1706.08500*.

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

Hinton, G. E., S. Osindero, and Y. Teh (2006). A fast learning algorithm for deep belief nets. *Neural Computation 18*, 1527–1554.

Hinton, G. E. and R. Salakhutdinov (2006, July). Reducing the dimensionality of data with neural networks. *Science 313*(5786), 504–507.

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

Hinton, G. E., O. Vinyals, and J. Dean (2014). Distilling the knowledge in a neural network. In *NIPS 2014 Deep Learning Workshop*.

Hornik, K., M. Stinchcombe, and H. White (1989). Multilayer feedforward networks are universal approximators. *2*, 359–366.

Hu, Z., Z. Yang, R. Salakhutdinov, and E. P. Xing (2017). On unifying deep generative models. *arXiv preprint arXiv:1706.00550*.

Hubel, D. H. and T. N. Wiesel (1959). Receptive fields of single neurons in the cat's striate cortex. *Journal of Physiology 148*, 574–591.

Hyvärinen, A. (2005). Estimation of non-normalized statistical models using score matching. *6*, 695–709.

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

Jacobs, R. A., M. I. Jordan, S. J. Nowlan, and G. E. Hinton (1991). Adaptive mixtures of local experts. *Neural computation 3*(1), 79–87.

Jarrett, K., K. Kavukcuoglu, M. Ranzato, and Y. LeCun (2009). What is the best multi-stage architecture for object recognition? In *Proc. International Conference on Computer Vision (ICCV'09)*, pp. 2146–2153. IEEE.

Jin, X., C. Xu, J. Feng, Y. Wei, J. Xiong, and S. Yan (2016). Deep learning with s-shaped rectified linear activation units.

Jolliffe, I. T. (1986). *Principal Component Analysis*. New York: Springer-Verlag.

Kim, T. and Y. Bengio (2016). Deep directed generative models with energy-based probability estimation. *arXiv preprint arXiv:1606.03439*.

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

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

Kodali, N., J. Abernethy, J. Hays, and Z. Kira (2017). How to train your dragan. *arXiv preprint arXiv:1705.07215*.

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

Krizhevsky, A., I. Sutskever, and G. Hinton (2012). ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25 (NIPS'2012)*.

Kullback, S. and R. A. Leibler (1951). On information and sufficiency. *Annals of Mathematical Statistics 22*, 49–86.

Larochelle, H. and I. Murray (2011). The Neural Autoregressive Distribution Estimator. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (AISTATS'2011)*, Volume 15 of JMLR: W&CP.

Larsen, A. B. L., S. K. Sønderby, and O. Winther (2015). Autoencoding beyond pixels using a learned similarity metric. *CoRR abs/1512.09300*.

Le, Q., M. Ranzato, R. Monga, M. Devin, G. Corrado, K. Chen, J. Dean, and A. Ng (2012). Building high-level features using large scale unsupervised learning. In *ICML'2012*.

LeCun, Y. (1989). Generalization and network design strategies. In *Connectionism in Perspective*. Elsevier Publishers.

LeCun, Y., L. Bottou, Y. Bengio, and P. Haffner (1998, November). Gradient-based learning applied to document recognition. *Proceedings of the IEEE 86*(11), 2278–2324.

Lee, D.-H., S. Zhang, A. Fischer, and Y. Bengio (2015). Difference target propagation. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 498–515. Springer.

Li, W., R. Zhao, T. Xiao, and X. Wang (2014). Deepreid: Deep filter pairing neural network for person re-identification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 152–159.

Liu, Z., P. Luo, X. Wang, and X. Tang (2015). Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*.

Maas, A. L., A. Y. Hannun, and A. Y. Ng (2013). Rectifier nonlinearities improve neural network acoustic models. In *ICML Workshop on Deep Learning for Audio, Speech, and Language Processing*.

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

Malinowski, M. and M. Fritz (2013). Learnable pooling regions for image classification. In *International Conference on Learning Representations: Workshop track*.

Martens, J. (2010, June). Deep learning via Hessian-free optimization. pp. 735–742.

Mikolov, T., K. Chen, G. Corrado, and J. Dean (2013). Efficient estimation of word representations in vector space. In *International Conference on Learning Representations: Workshops Track*.

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

Mnih, A. and K. Gregor (2014). Neural variational inference and learning in belief networks. In *ICML'2014*.

Nair, V. and G. E. Hinton (2010). Rectified linear units improve restricted Boltzmann machines. pp. 807–814.

Neal, R. M. (2001, April). Annealed importance sampling. *Statistics and Computing 11*(2), 125–139.

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

Ng, A. Y., M. I. Jordan, and Y. Weiss (2002). On spectral clustering: analysis and an algorithm. Cambridge, MA. MIT Press.

Nguyen, A., J. Yosinski, and J. Clune (2015). Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 427–436. IEEE.

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

Opitz, D. and R. Maclin (1999). Popular ensemble methods: An empirical study. *Journal of Artificial Intelligence Research 11*, 169–âĂŞ198.

Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann.

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

Raina, R., A. Madhavan, and A. Y. Ng (2009). Large-scale deep unsupervised learning using graphics processors. New York, NY, USA, pp. 873–880.

Reed, S. E., Z. Akata, X. Yan, L. Logeswaran, B. Schiele, and H. Lee (2016). Generative adversarial text to image synthesis. *CoRR abs/1605.05396*.

Rezende, D. J., S. Mohamed, and D. Wierstra (2014). Stochastic backpropagation and approximate inference in deep generative models. In *ICML'2014*.

Rifai, S., Y. Dauphin, P. Vincent, Y. Bengio, and X. Muller (2011). The manifold tangent classifier. In *NIPS'2011*. Student paper award.

Rosca, M., B. Lakshminarayanan, D. Warde-Farley, and S. Mohamed (2017). Variational approaches for auto-encoding generative adversarial networks. *arXiv preprint arXiv:1706.04987*.

Rumelhart, D. E., G. E. Hinton, and R. J. Williams (1986). Learning representations by back-propagating errors. *Nature 323*, 533–536.

Russakovsky, O., J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei (2014). ImageNet Large Scale Visual Recognition Challenge.

Salakhutdinov, R. and G. Hinton (2009). Deep Boltzmann machines. In *Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics (AISTATS 2009)*, Volume 8.

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

Salinas, E. and L. F. Abbott (1996, October). A model of multiplicative neural responses in parietal cortex. *Proc Natl Acad Sci U S A 93*(21), 11956–11961.

Saul, L. K. and M. I. Jordan (1996). Exploiting tractable substructures in intractable networks. MIT Press, Cambridge, MA.

Schapire, R. E. (1990). The strength of weak learnability. *Machine Learning 5*(2), 197–227.

Schraudolph, N. N. (1998). Centering neural network gradient factors. In G. B. Orr and K.-R. Muller (Eds.), *Neural Networks: Tricks of the Trade*, pp. 548–548. Springer.

Schroff, F., D. Kalenichenko, and J. Philbin (2015). Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 815–823.

Sermanet, P., S. Chintala, and Y. LeCun (2012). Convolutional neural networks applied to house numbers digit classification. In *International Conference on Pattern Recognition (ICPR 2012)*.

Sermanet, P., D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun (2014, April). Overfeat: Integrated recognition, localization and detection using convolutional networks. In *International Conference on Learning Representations (ICLR2014)*. CBLS.

Shachter, R. D. (1998). Bayes-ball: The rational pastime (for determining irrelevance and requisite information in belief networks and influence diagrams). pp. 480–487.

Shazeer, N., A. Mirhoseini, K. Maziarz, A. Davis, Q. Le, G. Hinton, and J. Dean (2017). Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*.

Smirnov, E. (2013). North atlantic right whale call detection with convolutional neural networks.

Snoek, J., H. Larochelle, and R. P. Adams (2012). Practical bayesian optimization of machine learning algorithms. In *Neural Information Processing Systems*.

Sønderby, C. K., J. Caballero, L. Theis, W. Shi, and F. Huszár (2016). Amortised map inference for image super-resolution. *arXiv preprint arXiv:1610.04490*.

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

Springenberg, J. T., A. Dosovitskiy, T. Brox, and M. Riedmiller (2015). Striving for simplicity: The all convolutional net. In *ICLR*.

Srebro, N. and A. Shraibman (2005). Rank, trace-norm and max-norm. In *Proceedings of the 18th Annual Conference on Learning Theory*, pp. 545–560. Springer-Verlag.

Srivastava, N. (2013). Improving neural networks with dropout. Master's thesis, U. Toronto.

Sutskever, I., R. Józefowicz, K. Gregor, D. J. Rezende, T. Lillicrap, and O. Vinyals (2015). Towards principled unsupervised learning. *CoRR abs/1511.06440*.

Sutskever, I., J. Martens, G. Dahl, and G. Hinton (2013). On the importance of initialization and momentum in deep learning. In *ICML*.

Swietojanski, P., J. Li, and J.-T. Huang (2014). Investigation of maxout networks for speech recognition. In *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*, pp. 7649–7653. IEEE.

Szegedy, C., W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich (2015). Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1–9.

Szegedy, C., V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna (2015, December). Rethinking the Inception Architecture for Computer Vision. *ArXiv e-prints*.

Szegedy, C., W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. J. Goodfellow, and R. Fergus (2014). Intriguing properties of neural networks. *ICLR abs/1312.6199*.

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

Valiant, L. G. (1984). A theory of the learnable. *Communications of the ACM 27*(11), 1134–1142.

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

Variani, E., X. Lei, E. McDermott, I. L. Moreno, and J. Gonzalez-Dominguez (2014). Deep neural networks for small footprint text-dependent speaker verification. In *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*, pp. 4052–4056. IEEE.

Vincent, P. (2011, July). A connection between score matching and denoising autoencoders. *Neural Computation 23*(7), 1661–1674.

Vincent, P., H. Larochelle, Y. Bengio, and P.-A. Manzagol (2008). Extracting and composing robust features with denoising autoencoders. In W. W. Cohen, A. McCallum, and S. T. Roweis (Eds.), *Proceedings of the Twenty-fifth International Conference on Machine Learning (ICML'08)*, pp. 1096–1103. ACM.

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

Wager, S., S. Wang, and P. Liang (2013). Dropout training as adaptive regularization. In *Advances in Neural Information Processing Systems 26*, pp. 351–359.

Wan, L., M. Zeiler, S. Zhang, Y. LeCun, and R. Fergus (2013). Regularization of neural networks using dropconnect. In *ICML'2013*.

Wang, S. (2004). General constructive representations for continuous piecewise-linear functions. *IEEE Trans. Circuits Systems 51*(9), 1889–1896.

Wang, S. and C. Manning (2013). Fast dropout training. In *ICML'2013*.

Warde-Farley, D. and Y. Bengio (2017). Improving generative adversarial networks with denoising feature matching. In *ICLR 2017*.

Warde-Farley, D. and I. Goodfellow (2016). Adversarial perturbations of deep neural networks. In T. Hazan, G. Papandreou, and D. Tarlow (Eds.), *Perturbation, Optimization and Statistics*, Chapter 11, pp. 311–339. Cambridge: MIT Press.

Warde-Farley, D., I. Goodfellow, A. Courville, and Y. Bengio (2014). An empirical analysis of dropout in piecewise linear networks. In *ICLR 2014*.

Warde-Farley, D., A. Rabinovich, and D. Anguelov (2015). Self-informed neural network structure learning. In *ICLR 2015, Workshop Track*.

Wilcoxon, F. (1945, December). "individual comparisons by ranking methods. *Biometrics Bulletin 1*(6), 80–83.

Williams, R. J. (1992). Simple statistical gradient-following algorithms connectionist reinforcement learning. *Machine Learning 8*, 229–256.

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

Yosinski, J., J. Clune, Y. Bengio, and H. Lipson (2014). How transferable are features in deep neural networks? In *Advances in neural information processing systems*, pp. 3320–3328.

Yosinski, J., J. Clune, A. Nguyen, T. Fuchs, and H. Lipson (2015). Understanding neural networks through deep visualization. *arXiv preprint arXiv:1506.06579*.

Yu, D. and L. Deng (2011). Deep convex net: A scalable architecture for speech pattern classification. In *INTERSPEECH*, pp. 2285–2288.

Yu, F., Y. Zhang, S. Song, A. Seff, and J. Xiao (2015). LSUN: construction of a large-scale image dataset using deep learning with humans in the loop. *CoRR abs/1506.03365*.

Zeiler, M. D. and R. Fergus (2013). Stochastic pooling for regularization of deep convolutional neural networks. In *International Conference on Learning Representations*.

Zeiler, M. D., M. Ranzato, R. Monga, M. Mao, K. Yang, Q. Le, P. Nguyen, A. Senior, V. Vanhoucke, J. Dean, and G. E. Hinton (2013). On rectified linear units for speech processing. In *ICASSP 2013*.

Zhang, C., S. Bengio, M. Hardt, B. Recht, and O. Vinyals (2016). Understanding deep learning requires rethinking generalization. *arXiv preprint arXiv:1611.03530*.

Zhang, X., J. Trmal, D. Povey, and S. Khudanpur (2014). Improving deep neural network acoustic models using generalized maxout networks. In *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*, pp. 215–219. IEEE.

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

Zoph, B., V. Vasudevan, J. Shlens, and Q. V. Le (2017). Learning transferable architectures for scalable image recognition. *arXiv preprint arXiv:1707.07012*.