Université de Montréal

Reparametrization in Deep Learning

par Laurent Dinh

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

Février, 2018

Résumé

L'apprentissage profond est une approche connectioniste à l'apprentissage automatique. Elle a pu exploiter la récente production massive de données numériques et l'explosion de la quantité de ressources computationelles qu'a amené ces dernières décennies. La conception d'algorithmes d'apprentissage profond repose sur trois facteurs essentiels: l'expressivité, la recherche efficace de solution, et la généralisation des solutions apprises. Nous explorerons dans cette thèse ces thèmes du point de vue de la reparamétrisation.

Plus précisement, le chapitre 3 s'attaque à une conjecture populaire, selon laquelle les énormes réseaux de neurones ont pu apprendre, parmi tant de solutions possibles, celle qui généralise parce que les minima atteints sont *plats*. Nous démontrons les lacunes profondes de cette conjecture par reparamétrisation sur des exemples simples de modèles populaires, ce qui nous amène à nous interroger sur les interprétations qu'ont superposées précédents chercheurs sur plusieurs phénomènes précédemment observés.

Enfin, le chapitre 5 enquête sur le principe d'analyse non-linéaire en composantes indépendantes permettant une formulation analytique de la densité d'un modèle par changement de variable. En particulier, nous proposons l'architecture Real NVP qui utilise de puissantes fonctions paramétriques et aisément inversible que nous pouvons simplement entraîner par descente de gradient. Nous indiquons les points forts et les points faibles de ce genre d'approches et expliquons les algorithmes développés durant ce travail.

Mots clés: réseaux de neurones, réseaux neuronaux, réseaux de neurones profonds, réseaux neuronaux profonds, apprentissage automatique, apprentissage profond, apprentissage non-supervisé, modélisation probabiliste, modélisation générative, modèles probabilistes, modèles génératifs, réseaux générateurs, inférence variationnelle, généralisation, astuce de la reparamétrisation

Summary

Deep learning is a connectionist approach to machine learning that successfully harnessed our massive production of data and recent increase in computational resources. In designing efficient deep learning algorithms come three principal themes: expressivity, trainability, and generalizability. We will explore in this thesis these questions through the point of view of reparametrization.

In particular, chapter 3 confronts a popular conjecture in deep learning attempting to explain why large neural network are learning among many plausible hypotheses one that generalize: *flat minima* reached through learning generalize better. We demonstrate the serious limitations this conjecture encounters by reparametrization on several simple and popular models and interrogate the interpretations put on experimental observations.

Chapter 5 explores the framework of nonlinear independent components enabling closed form density evaluation through change of variable. More precisely, this work proposes Real NVP, an architecture using expressive and easily invertible computational layers trainable by standard gradient descent algorithms. We showcase its successes and shortcomings in modelling high dimensional data, and explain the techniques developed in that design.

Keywords: neural networks, deep neural networks, machine learning, deep learning, unsupervised learning, probabilistic modelling, probabilistic models, generative modelling, generative models, generator networks, variational inference, generalization, reparametrization trick

Contents

List of Figures

List of Tables

List of Abbreviations

AdaM	Adaptative Moment estimation
AE	Auto-Encoder
CDF	Cumulative Distribution Function
DNN	Deep Neural Network
CNN	Convolutional Neural Network
ELBO	Evidence Lower BOund
GD	Gradient Descent
GMM	Gaussian Mixture Model
HMM	Hidden Markov Model
KL Divergence	Kullback-Liebler Divergence
KLD	Kullback-Liebler Divergence
LISA	Laboratoire d'Informatique des Systèmes Adaptifs
LSTM	Long-Short Term Memory
MoG	Mixture of Gaussians
MADE	Masked Autoencoder for Density Estimation
MCMC	Markov Chain Monte Carlo
MDL	Minimum Description Length
MILA	Montréal Institute of Learning Algorithms
MLE	Maximum Likelihood Estimation
MLP	Multi-Layer Perceptron
MSE	Mean Squared Error
NADE	Neural Autoregressive Density Estimator
NICE	Nonlinear Independent Components Estimation
NLL	Negative Log-Likelihood
PAC	Probably Approximately Correct
PMF	Probability Mass Function
PDF	Probability Density Function
Real NVP	Real-valued Non-Volume Preserving
RNN	Recurrent Neural Network
SGD	Stochastic Gradient Descent
VAE	Variational Auto-Encoder
XE	Cross Entropy

Acknowledgments

> you will take the long way to get to these Orions. the long way will become a theme in your life, but a journey you learn to love.
>
> Solange Knowles (2017)

This doctoral program has been a tremendous opportunity to work on a research topic I was passioned about in the blooming field of deep learning. Pursuing this endeavor would not have been possible without the consistent support of my family, friends, colleagues, and advisors.

I want to thank Professor and CIFAR Senior Fellow Yoshua Bengio for communicating his passion for artificial intelligence, welcoming in and guiding me through an internship and a doctoral program, and for his patience and support, which allowed me to set up my own research agenda and style, and learn to manage community expectation and my own.

Following the impressive growth of deep learning, the lab I was navigating in underwent a radical structural transformation. I want to thank organizers of this lab, Céline Bégin, Linda Penthière, Angela Fahdi, Jocelyne Étienne, and Myriam Côté, for making this growth possible. I also want to thank friends, collaborators, co-workers, and colleagues who contributed to the different phases *MILA* (formerly known as *LISA*) underwent, with whom I had the pleasure to have fruitful interactions, in terms of work or learning experience, who provided moral support, and who shared their enthusiasm for their craft, including: Aaron Courville, Adriana Romero, Adrien Ali Taïga, Ahmed Touati, Alexandre De Brébisson, Amar Shah, Amjad Almahairi, Amy Zhang, Anna Huang, Antti Rasmus, Arnaud Bergeron, Asja Fisher, Atousa Torabi, Bart van Merriënboer, Çağlar Gülçehre, César Laurent, Chiheb Trabelsi, Chin-Wei Huang, Daniel Jiwoong Im, David Krueger, David Warde-Farley, Devon Hjelm, Dmitriy Serdyuk, Dustin Webb,

Dzmitry Bahdanau, Eric Laufer, Faruk Ahmed, Felix Hill, Francesco Visin, Frédéric Bastien, Gabriel Huang, Grégoire Mesnil, Guillaume Alain, Guillaume Desjardins, Harm De Vries, Harri Valpola, Ian Goodfellow, Ioannis Mitliagkas, Ishaan Gulrajani, Jacob Athul Paul, Jessica Thompson, João Felipe Santos, Jörg Bornschein, José Sotelo, Junyoung Chung, Kelvin Xu, Kratarth Goel, Kyunghyun Cho, Kyle Kastner, Li Yao, Marcin Moczulski, Mathias Berglund, Mathieu Germain, Mehdi Mirza, Mélanie Ducoffe, Mohammad Pezeshski, Negar Rostamzadeh, Nicolas Ballas, Orhan Firat, Pascal Lamblin, Pascal Vincent, Razvan Pascanu, Saizheng Zhang, Samira Shabanian, Simon Jégou, Simon Lefrançois, Simon Lacoste-Julien, Sungjin Ahn, Tapani Raiko, Tegan Maharaj, Tim Cooijmans, Thomas Le Paine, Valentin Bisson, Vincent Dumoulin, Xavier Glorot, Yann Dauphin, Yaroslav Ganin, and Ying Zhang.

My visits in different research groups (machine learning group in University of British Columbia, Google Brain, and DeepMind) also provided me with valuable opportunities to learn and build a more unique research agenda. Thanks to Samy Bengio, Nando de Freitas, and Helen King for inviting me in their research groups and providing supervision. I am also grateful to the friends, collaborators, co-workers, and colleagues I met during these visits or in conferences. They provided productive collaborations, welcomed me in their group, shared valuable insights and experiences, or inspired by their contribution, among them: Aäron van den Oord, Andrew Dai, Anelia Angelova, Angeliki Lazaridou, Ariel Herbert-Voss, Arvind Neelakantan, Aurko Roy, Babak Shakibi, Ben Poole, Bobak Shahriari, Brandon Amos, Brendan Shillingford, Brian Cheung, Charles Frye, Chelsea Finn, Danielle Belgrave, Danilo Jimenez Rezende, David Andersen, David Grangier, David Ha, David Matheson, Deirdre Quillen, Diane Bouchacourt, Durk Kingma, Édouard Oyallon, Edward Grefenstette, Emmy Qin, Eric Jang, Eugene Belilovsky, Gabriel Synnaeve, Gabriella Contardo, Georgia Gkioxari, George Edward Dahl, Grzegorz Swirszcz, Hugo Larochelle, Ilya Sutskever, Irwan Bello, Jackie Kay, Jamie Kiros, Jascha Sohl-Dickstein, Jean Pouget-Abadie, Johanna Hansen, Jon Gauthier, Jon Shlens, Joost Van Amersfoort, Junhyuk Oh, Justin Bayer, Karol Hausman, Konstantinos Bousmalis, Luke Vilnis, Maithra Raghu, Marc Gendron Bellemare, Mareija Baya, Matthew Hoffman, Matthew W. Hoffman, Max Welling, Mengye Ren, Mike Schuster,

Misha Denil, Mohammad Norouzi, Moustapha Cissé, Naomi Saphra, Nal Kalchbrenner, Natasha Jaques, Navdeep Jaitly, Neha Wadia, Nicolas Le Roux, Nicolas Usunier, Nicole Rafidi, Olivier Pietquin, Oriol Vynials, Peter Xi Chen, Pierre Sermanet, Pierre-Antoine Manzagol, Prajit Ramachandran, Quoc Viet Le, Rafal Jozefowicz, Rahul Krishnan, Raia Hadsell, Rémi Munos, Richard Socher, Rupesh Srivastava, Sam Bowman, Sander Dieleman, Scott Reed, Sergio Gómez Colmenarejo, Shakir Mohamed, Sherry Moore, Simon Osindero, Soham De, Stephan Gouws, Stephan Zheng, Shixiang Shane Gu, Taesung Park, Takeru Miyato, Tejas Kulkarni, Théophane Weber, Tim Salimans, Timnit Gebru, Tina Zhu, William Chan, Wojciech Zaremba, Yannis Assael, Yutian Chen, Yuke Zhu, Zhongwen Xu, and Ziyu Wang.

I thank as well my teachers who prepared me for this program: Arnak Dalalyan, Erick Herbin, Gábor Lugosi, Gilles Faÿ, Guillaume Obozinski, Iasonas Kokkinos, Igor Kornev, Jean-Philippe Vert, Laurent Cabaret, Laurent Dumas, Lionel Gabet, Matthew Blaschko, Michael I. Jordan, Nicolas Vayatis, Nikos Paragios, Patrick Teller, Pauline Lafitte, Pawan Kumar, and Rémi Munos.

I am also grateful for the suport of my family, not only my parents I am in debt to but also my siblings Huy, Hoà, and Laura, and my cousins, including Isabelle, Lucie, Laura, and Arthur.

Thanks to my friends who kept me grounded and without whom I wouldn't be here, particularly: Alexandre, Ansona, Antoine, Amanda, Amin, Audrey, Baptiste, Charlotte, Claire, Diane, Fabien, Florence, Guillaume, Hadrien, Ibrahima, Jean-Charles, Jessica, Johanna, Loic, Marguerite, Martin, Mathilde, Nicolas, Nicole, Rebecca, Sévan, Tarik, Thomas, and Vincent.

Thank you Nicolas Chapados for providing the thesis template, and Kyle Kastner and Junyoung Chung for showing me how to use it. Additionally, I'm grateful to Junyoung Chung, Adriana Romero, Harm De Vries, Kyle Kastner and Édouard Oyallon for providing feedback to this dissertation.

Finally, the work reported in this thesis would not have been possible without the financial support from: Ubisoft, NSERC, Calcul Quebec, Compute Canada, the Canada Research Chairs and CIFAR.

1 Machine learning

Computer science can be defined as the study and application of efficient automatic algorithmic processes. This field has known unreasonable effectiveness in its application to domains like commerce, finance, entertainment, logistics, and education through by enabling the automation of simple tasks such as querying, storing, and processing data at impressive scale.

However, most of these algorithmic processes were traditionally implemented through direct programming. As direct programming involved careful writing of explicit handcrafted rules, logics, and heuristics, researchers soon realized that such traditional approach proved to be too cumbersome for tasks like *computer vision* and *natural language processing* to be efficient. Indeed, although these problems were mostly trivial for the average neurotypical, able-bodied person, manually designing good solutions was near impossible.

The work presented in this thesis has been done in the framework of *machine learning*. Machine learning is a subfield of computer science that aimed to address this shortcoming in direct programming. Instead, it would enable to specify non-constructively a solution to these problems through examples of correct behaviors. The model would then learn on that data to obtain the desired behavior.

The recent increase in data production and computational power that happened in the last few decades was a key component in the success of this approach and appropriately addressed new challenges in information processing and automation brought by this surge, now making machine learning an important, if not essential, part of computer science.

I will introduce machine learning briefly in this chapter but further details on the concept can be found in Bishop (2006); Murphy (2012); Goodfellow et al. (2016).

1.1 Definition

Machine learning is the study of systems that can adapt and learn from data. This paradigm becomes particularly useful when collecting a large amount of data on these complex tasks can prove to be more productive and less human labor intensive than hand-crafting programmed behaviors. Relying instead on increasing computing power, machine learning would, in this situation, harness this large amount of data to induce a set of behaviors to solve these problems.

More formally, machine learning is characterized by the ability to "learn from experience E with respect to some class of tasks T and performance measure P", i.e. "its performance at tasks in T, as measured by P, improves with experience E" (Mitchell, 1997). Since the said experience, tasks, and performance measure largely vary depending on the problem at hand, I will specify and detail the domains we are interested in.

Experience The nature of the experience of a machine learning algorithm will be numerical. Often the numbers presented to this algorithm will take the form of a set of elements in an input domain $\mathcal{X}$, which is usually finite sequences of scalars or vectors. The domain can be a finite set of elements, scalar $\mathbb{R}$, vector space $\mathbb{R}^{d_\mathcal{X}}$ (with $d_\mathcal{X}$ the data dimensionality), finite sequences of these domains $\bigcup_{m\in\mathbb{N}^+} (\mathcal{X}_{sub})^m$, or any combination of these domains. Most commonly, the base domain will be a vector space $\mathbb{R}^{d_{\mathcal{X}_{sub}}}$, each element of a vector of that domain are called a *feature*. In *supervised learning*, this domain is supplemented with an output domain $\mathcal{Y}$.

A common assumption on these series is that they are *independently and identically distributed*, meaning each *example* is drawn independently from the same data distribution p^*_X. This assumption is often essential in establishing the generalization capability of most machine learning algorithms.

Task Machine learning algorithms can be used on a large array of problems. These problems are often reframed in a more abstract task including *supervised learning* and *unsupervised learning* tasks. Supervised learning tasks are characterized by its goals of finding a mapping f from input space $\mathcal{X}$ to output space $\mathcal{Y}$. The output space for *classification*, *regression*, and *structured prediction* are

respectively a finite set, scalar, or a subset of finite sequences of these domains. Unsupervised learning aims at characterizing the input space distribution itself in some meaningful way, including *density estimation*, *clustering*, or *imputation of missing values*.

Performance measure One key component in describing a problem in machine learning is the performance measure we aim to optimize. More often, we minize instead its opposite, the loss function $\mathcal{L}(f)$. In our case, the loss function can be decomposed into a corresponding value $\ell(f, x)$ for each element $x \in \mathcal{X}$ (or $\ell(f, x, y)$ for each pair $(x, y) \in \mathcal{X} \times \mathcal{Y}$), resulting in an expression of the loss as the expectation

$$\mathcal{L}(f, p^*_X) = \mathbb{E}_{x \sim p^*_X}[\ell(f, x)] = \int_{x \in \mathcal{X}} \ell(f, x)\, dp^*_X(x)$$
$$\textit{or } \mathcal{L}(f, p^*_{X,Y}) = \mathbb{E}_{x,y \sim p^*_{X,Y}}[\ell(f, x, y)] = \int_{(x,y) \in \mathcal{X} \times \mathcal{Y}} \ell(f, x, y)\, dp^*_{X,Y}(x, y).$$

A common loss function in classification is for example the 0-1 *loss*: $\ell_{01}(f, x, y) = \mathbb{1}\left(f(x) = y\right)$.

Given these definitions, the task of *learning* consists in finding a good, if not optimal, behavior among a set $\mathcal{F}$ of possible functions, called the *hypothesis space*, in order to minimize this *expected loss* $\mathcal{L}(f, p^*_X) = \mathbb{E}_{x \sim p^*_X}[\ell(f, x)]$, also called the *expected risk*.

However, this data distribution over which the expectation would be computed is not directly provided to the system. Instead, the system is only provided with a finite amount of data $\mathcal{D} = (x^{(n)})_{n \leq N} \in \mathcal{X}^N$ that will allow the model to properly infer a correct behavior. One way to find an approximate solution is *empirical risk minimization* (Vapnik, 1992), where we minimize a Monte Carlo approximation of the expected loss called the *empirical loss* (also known as *empirical risk*, Vapnik, 2013),

$$\hat{\mathcal{L}}(f, \mathcal{D}) = \frac{1}{N} \sum_{n=1}^{N} \ell(f, x^{(n)}).$$

This principle serves as the basis of reformulating machine learning as an optimiza-

tion in the hope that this minimization will result in a reasonable solution with respect to the expected loss, which we also call the *generalization error*. This generalization loss is also estimated by Monte Carlo using separate data from the same distribution[1]. To this effect, the data is often separated in at least two subsets, the *training subset* $\mathcal{D}_{\text{train}}$, on which the algorithm is trained, and the *test subset* $\mathcal{D}_{\text{test}}$, on which the expected loss is estimated.

1.2 Examples

1.2.1 Linear regression

One of the simplest examples of machine learning problem is the scalar *linear regression* problem. As the name indicates, the class of functions used will be the set of linear functions from $\mathbb{R}^{d_\mathcal{X}} \mapsto \mathbb{R}$ (see figure 1.1b),

$$\mathcal{F}_{linear} = \left\{ f : x = (x_i)_{i \leq d_\mathcal{X}} \mapsto \sum_{i=1}^{d_\mathcal{X}} w_i \cdot x_i, \forall w = (w_i)_{i \leq d_\mathcal{X}} \in \mathbb{R}^{d_\mathcal{X}} \right\}.$$

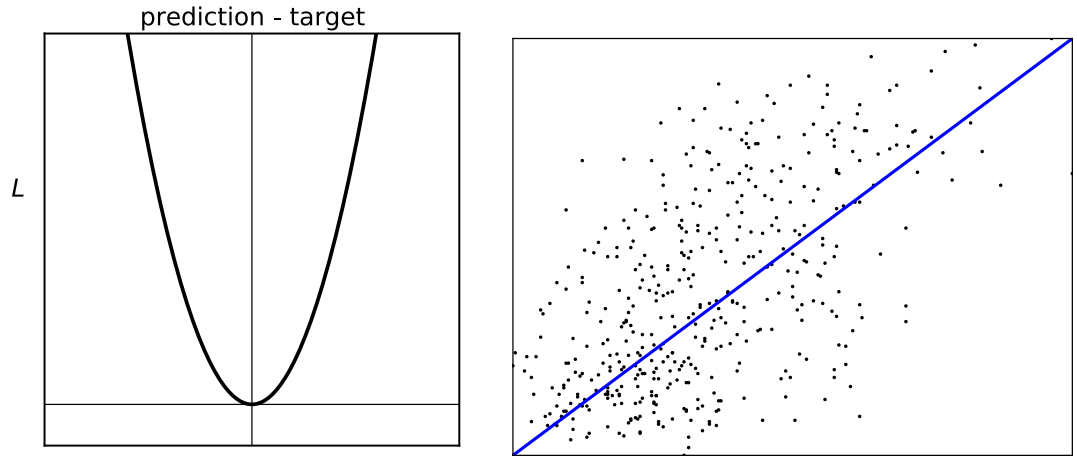
In reality, we are considering the set of affine functions in $\mathbb{R}^{d_\mathcal{X}} \mapsto \mathbb{R}$, which would be equivalent to linear functions on the augmented vectors $\tilde{x} = (1, x_0, \ldots, x_{d_\mathcal{X}})$.

Let $\mathcal{D} = (x^{(n)}, y^{(n)})_{n \leq N} \in (\mathbb{R}^{d_\mathcal{X}}, \mathbb{R})^N$, the loss we aim to minimize in this regression problem is the *mean squared loss* (see figure 1.1a)

$$\hat{\mathcal{L}}_{MSE}(f, \mathcal{D}) = \frac{1}{N} \sum_{n=1}^{N} \left(f(x^{(n)}) - y^{(n)} \right)^2$$
$$\hat{\mathcal{L}}_{MSE}(w, \mathcal{D}) = \frac{1}{N} \sum_{n=1}^{N} (< x^{(n)}, w > - y^{(n)})^2.$$

Although the linear function hypothesis space is so limited that it would rarely contain the data generating behavior in most realisitic cases, it allows for closed form solution inference. Let X be the *design matrix*, a matrix of size $n \times d$ whose

[1] We don't consider in this dissertation the important case of *transfer learning* when the objective is loss minimization on another distribution.



(a) Plot of the quadratic loss with respect to the difference between the prediction and the target.

(b) A 2-D visualization of a trained linear regression model with the associated data points. The learned relationship between input and output is constrained to be affine. In some cases (including this one), that it enough.

Figure 1.1 – Visualization of a trained linear regression model with the associated loss function.

rows are corresponding to the vectors $(x^{(n)})_{n\leq N}$, and $y \in \mathbb{R}^N$ the vector containing each corresponding value $(y^{(n)})_{n\leq N}$. The loss function can be rewritten

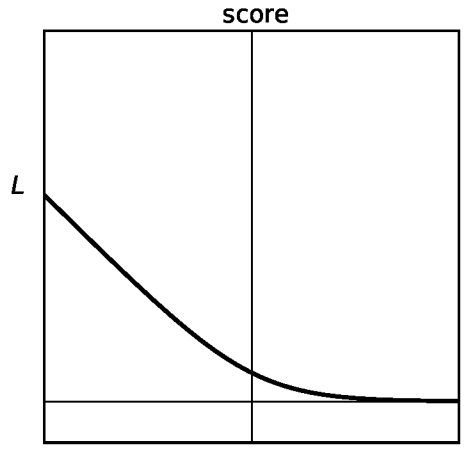
$$\hat{\mathcal{L}}_{MSE}(w, \mathcal{D}) = \frac{1}{N}(Xw^T - y^T)^T(Xw^T - y^T)$$

which is minimized when its derivative with respect to w is zero

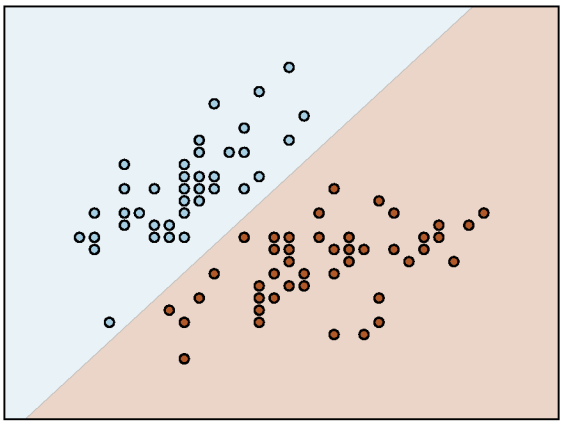
$$\frac{\partial \hat{\mathcal{L}}_{MSE}(w, \mathcal{D})}{\partial w^T} = \frac{1}{N}(Xw^T - y^T) = 0 \Rightarrow X^T X w^T = X^T y^T \Rightarrow w^T = (X^T X)^{-1} X^T y^T.$$

1.2.2 Logistic regression

Despite its name, *logistic regression* is a classification problem, meaning that the data of interest $\mathcal{D} = (x^{(n)}, y^{(n)})_{n\leq N}$ is in $(\mathbb{R}^{d_X}, \{0, 1\})^N$ for the binary case. The class of function is still $\mathcal{F}_{linear}$ but instead of predicting values directly the predictor will output a score for "positive" label 1, we then follow the *plug-in rule*: if the score is positive then the prediction will be 1, otherwise it will be 0.



(a) Plot of the logitic loss with respect to the score of the matching class.



(b) A 2-D visualization of a trained logistic regression model with the associated data points. The decision boundary between the two classes is constrained to be linear. In some cases (including this one), that it enough.

Figure 1.2 – Visualization of a trained binary logistic regression model with the associated loss function. Best seen with colors.

The loss function used will be the *logistic loss* (see figure 1.2a)

$$\hat{\mathcal{L}}_{log}(f, \mathcal{D}) = \frac{1}{N} \sum_{n=1}^{N} y^{(n)} \log(1 + e^{-f(x^{(n)})}) + (1 - y^{(n)}) \log(1 + e^{f(x^{(n)})})$$

$$\hat{\mathcal{L}}_{log}(w, \mathcal{D}) = \frac{1}{N} \sum_{n=1}^{N} y^{(n)} \log(1 + e^{-<x^{(n)}, w>}) + (1 - y^{(n)}) \log(1 + e^{<x^{(n)}, w>}).$$

This loss does not have a closed form solution for w, it is however *convex*, meaning that several standard optimization techniques can be applied to this problem with strong guarantees on optimization errors.

1.3 Tradeoffs of machine learning

The linear and logistic regression methods described above have the advantage of being fast and offering either closed form solution or applicable convex opti-

mization techniques. Unfortunately, both assume some degree of linearity in the problem, which is a very limited case. These algorithms are lacking expressive power.

A decomposition of the generalization error in Bottou and Bousquet (2008) provides insight on the desiderata of machine learning approaches. Let's consider an hypothesis space $\mathcal{F}$, $\mathcal{A}$ a stochastic algorithm which take a dataset $\mathcal{D}$ as input to return a predictor $\bar{f} \in \mathcal{F}$ by approximate empirical risk minimization, $\hat{f} = \arg\min_{f \in \mathcal{F}} \left(\hat{\mathcal{L}}(f, \mathcal{D})\right) \in \mathcal{F}$ a solution to empirical risk minimization in the family $\mathcal{F}$, $f^*_{\mathcal{F}} = \arg\min_{f \in \mathcal{F}} \left(\mathcal{L}(f)\right)$ an optimal predictor with respect to the generalization error in the family $\mathcal{F}$, and $f^* = \arg\min_f \left(\mathcal{L}(f)\right)$ the optimal predictor for generalization (not necessarily in $\mathcal{F}$), then Bottou and Bousquet (2008) proposes the following decomposition:

$$\begin{aligned}\mathbb{E}\left[\mathcal{L}(\bar{f})\right] = &\mathbb{E}\left[\mathcal{L}(\bar{f}) - \mathcal{L}(\hat{f})\right] \\ &+ \mathbb{E}\left[\mathcal{L}(\hat{f}) - \mathcal{L}(f^*_{\mathcal{F}})\right] \\ &+ \mathbb{E}\left[\mathcal{L}(f^*_{\mathcal{F}}) - \mathcal{L}(f^*)\right] \\ &+ \mathbb{E}\left[\mathcal{L}(f^*)\right].\end{aligned}$$

$\mathbb{E}\left[\mathcal{L}(f^*)\right]$ is the irreducible error you can have in a problem and computing this quantity requires generally to find the solution f^*. In that decomposition, the other terms are: $\mathbb{E}\left[\mathcal{L}(f^*_{\mathcal{F}}) - \mathcal{L}(f^*)\right]$ is called the *approximation error*, $\mathbb{E}\left[\mathcal{L}(\bar{f}) - \mathcal{L}(\hat{f})\right]$ is the *optimization error*, and $\mathbb{E}\left[\mathcal{L}(\hat{f}) - \mathcal{L}(f^*_{\mathcal{F}})\right]$ is the *estimation error*. To follow the nomenclature of Raghu et al. (2017), these terms correspond to the *expressivity*, *trainability*, and *generalizability* of a model.

1.3.1 Expressivity

One necessary condition for meaningful learning is to be *flexible* enough to adapt to training data. As mentioned earlier, the previous linear models (logistic regression and linear regression, see figure 1.1b and 1.2b) are however unlikely to include the optimal predictor f^* for most interesting problems. If the problem is on the contrary very nonlinear, such model are likely to *underfit*, i.e. not being able to approximate well the solution regardless of the quantity of data and computation. It is in general the case that the hypothesis space picked in our machine learning

algorithm does not include f^* and therefore introduces a *bias* in the solutions found. Given that observation, we woud like to adopt a family of predictors that would include a good enough approximation of f^*.

This is often done by increasing the *model capacity* to approximate different functions. To that effect, one could increase the number of features when specifying the model, increase the expressivity of a kernel function in shallow learning, or increase the size of a neural network in deep learning. The more functions a model can approximate the more *powerful* we consider it.

1.3.2 Trainability

If the effectiveness of a machine learning model was solely defined by its expressivity, a simple solution would be to arbitrarily increase model capacity to minimize the approximation error. However, one of the hurdles practioners would face with that approach is the finite computational resources at hand.

Machine learning methods related to empirical risk minimization are generally doing a search of the hypothesis space in order to approximately minimize a loss function. The larger and complex this hypothesis space, the less effective that search will be. For example, loss functions involving neural networks are notoriously *non-convex* in general, making the use of a convex optimization algorithm less reliable. Practical restriction in the ability to fully explore the hypothesis space results in a decrease in the *effective capacity* of a model.

Inference in the chosen model has to be at least *tractable*, i.e. computable in reasonable time, making a naive approach to *memory-based learning* (or *instance-based learning*) hardly *scalable* with the size of the dataset. Therefore it is here essential for machine learning models to discover reliable patterns and redundancy in the data to reduce in order to decrease unnecessary *computation* or memory usage.

1.3.3 Generalizability

Another reason to discover these patterns and redundancy is that they also enable meaningful learning. Generalization is a central question in machine learning: if the model was only able to extract information about the training data but could not extrapolate to new examples, the predictor would be hardly useful. Aimlessly

increasing a model's effective capacity often results in discovering and focusing on coincidental spurious patterns, i.e. noise, from the training set on which we do empirical risk minimization. As a result, the estimation error increases. We call this effect *overfitting*. The solution obtained from training on the Monte Carlo estimate of the expected loss is said to have high *variance*: the predictor is more a function of the particular instantiation of the dataset than of the task itself. A more common quantity to measure overfitting is the difference between generalization loss and training loss, called the *generalization gap*,

$$\mathcal{L}(f) - \hat{\mathcal{L}}(f, \mathcal{D}_{\text{train}}).$$

Under scalability constraints, increasing the amount of data used, artificially *augmenting the data* if necessary, reduces that variance reliably. If our model is *consistent*, it will converge to the true model f^* by definition. But collecting more data by an order of magnitude can be very expensive and time-consuming.

A way to overcome the variance of the resulting predictor is to average an *ensemble* of several valid hypotheses. One could, for example, implement *boostrap aggregation* (Breiman, 1996) (also known as *bagging*) and train on a few instances of the same model on different subsets of the training set. *Bayesian inference* (Neal, 2012; Andrieu et al., 2003; Solomonoff, 1978) averages on every valid hypotheses reweighted by their relevance according to an a priori knowledge and the training set.

Under that same computation budget, limiting the computation can also prove to be effective in reducing overfitting, but possibly at the cost of increased bias. Reducing model capacity is an example of such approach. One could also reduce the effective capacity by adding stochasticity in model search or model inference.

The model search can also be guided towards simpler hypotheses. For example, one can augment empirical risk minimization with a *regularization term* or *penalty term*, i.e. a term that aims at effectively bridging the difference between the empirical risk and the expected risk, in a similar fashion as the classic *generalization bounds* from statistical learning theory, resulting in the *regularized empirical risk minimization* problem

$$\hat{\mathcal{L}}_{Reg}(f, \mathcal{D}_{\text{train}}) = \hat{\mathcal{L}}(f, \mathcal{D}_{\text{train}}) + \mathcal{R}(f).$$

We call *regularization* the process of reducing (effective) model capacity in order to increase generalizability.

Since using such regularization would hinder performance on the training set, the regularization needs either to be decided prior to any training or tuned using a held-out subset of the dataset. Since the test set $\mathcal{D}_{\text{test}}$ must remain unused until final evaluation (e.g. Dwork et al., 2015), one should use instead a third separate subset, which is traditionally called the *validation set* $\mathcal{D}_{\text{valid}}$.

1.3.4 Tradeoff

Although bias, computation, and variance can be arbitrarily separated to understand phenomena in machine learning, they remain ultimately entangled in a *bias-variance-computation tradeoff*. The figure 1.3 shows the *bias-variance tradeoff*, a subset of this bias-variance-computation tradeoff.

Ideally, we wish to have an algorithm which will perform well on all counts: the model must be expressive, tractable, and generalize well. Now the infamous *no free lunch theorem* (Wolpert, 1996) states that no general purpose algorithm can fulfill those desiderata. Therefore, to enable the design of efficient learning algorithms, one must aim at restricting the set of tasks they want to solve. The less exhaustive is this set, the more opportunities we would have to provide a model with the appropriate *inductive bias*.

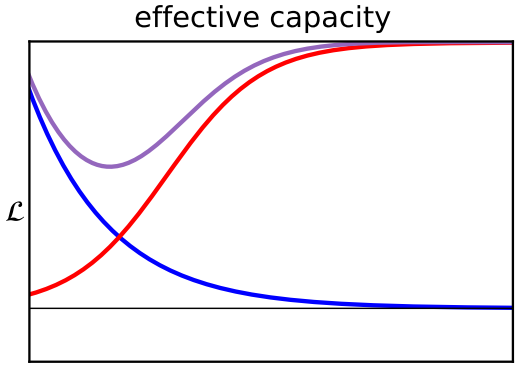
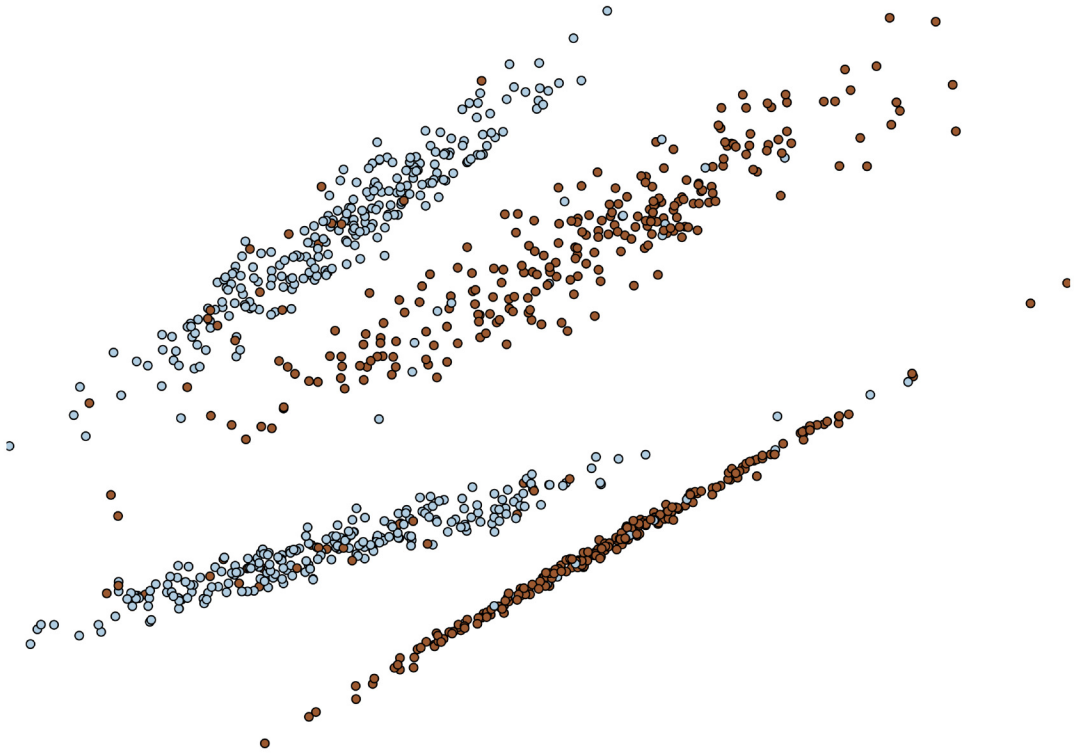
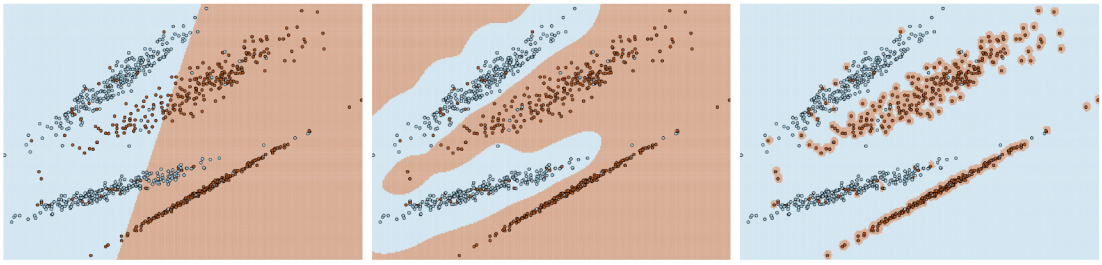


Figure 1.3 – A cartoon plot of the generalization loss with respect to the effective capacity. The blue line is the training loss, the red one is the generalization gap, and their sum, the generalization loss is in purple. This plot illustrate the *bias-variance tradeoff*: although increasing effective capacity decreases the training error, it increases the generalization gap. Although increasing effective capacity reduces training error, the return is decreasing exponentially. On the other hand, the incurred variance of the estimated solution $\hat{f}$ can increase with capacity to the point where the model is effectively useless. As a result, the generalization loss (in purple) is minimized in the middle. Best seen with colors.



(a) Classification problem, positively labeled points are in red and negatively labeled points are in blue. We see that although a nonlinear decision boundary can be approximately traced for this problem to separate the two classes, the classification dataset suffers from label noise, making the learning task more difficult.



(b) Underfitting classifier. The linear classification boundary cannot capture the complexity of the classification problem.

(c) Appropriate classifier. The nonlinear classification boundaries captures the complexity of the problem and the misclassified points were subject to label noise.

(d) Overfitting classifier. Although the nonlinear classification boundaries captures the complexity of the task, label noise results in spurious classification boundaries.

Figure 1.4 – Example of underfitting/overfitting in binary classification problem. Best seen with colors.

2 Deep learning

2.1 The hypothesis space

Bengio et al. (2007) define an *AI-set* of tasks of interest. These tasks includes perception, control, and planning. The raw sensory input involved in those tasks are typically high dimensional, e.g. a mere color image of size 64×64 already results in vectors of more than $10,000$ dimensions. As a a result, we are encountering a *curse of dimensionality* in these problems, when expressivity, trainability, and generalizability degrade exponentially with the number of dimensions.

A key ingredient in enabling learning on these problems is to exploit their regularities. A starting point for *local generalization* (i.e. around data points) is to embed the system with a *smoothness prior*: nearby points should have similar properties in terms of labeling and density[1]. This prior can be strengthened by taking advantage of the dominant structure of the data. In particular, one characteristic we hope to exploit in these tasks is the relatively *low intrinsic dimensionality* of the data. For instance, natural images remain around a data point predominantly along specific directions. This assumption is called the *manifold hypothesis*.

An efficient learning algorithm would hopefully be able to obtain *global generalization* (i.e. outside the immediate vicinity of data points). Whereas purely local template matching algorithms would resort to one-hot representations (an extremely sparse vector with only one non-zero component) *deep learning* (Bengio, 2009; Goodfellow et al., 2016) exploits the notion of *distributed representation* (Hinton, 1984), where information is instead distributed across multiple dimensions. An example of such distributed distribution would be a binary representation for a set of N integers, which takes less space than a one-hot binary representation ($\log_2(N)$ instead of N).

An ideal outcome would be to successfully disentangle factors of variation into

[1] In hindsight, the existence of adversarial examples (Szegedy et al., 2014; Goodfellow et al., 2015) in recent deep learning systems shows how weak such inductive bias is in those systems.

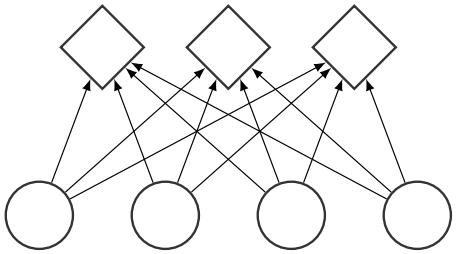


Figure 2.1 – Computational graph of a feedforward neural network layer. The computation goes upward, the botton nodes are the inputs and the top nodes are the outputs. Each output node take all the input with different strength (defined by the weight matrix), before applying a nonlinearity if necessary.

different features, which would allow an efficient reusability of these *disentangled features*. Such features facilitate global generalization by sharing statistical strength, by exploiting recombinations of the properties of training examples, one can extrapolate outside their vicinity.

2.1.1 Artificial neural networks

A key component in learning those distributed representations are artificial neural networks, composed typically from feedforward layers with functional equation

$$h = \phi_{act}(xW + b)$$

where h is a vector output, the result of an affine operation, represented by the matrix W, called the *weights*, and a vector b called the *offset*, composed with an element-wise *activation function* ϕ_{act}, which can be

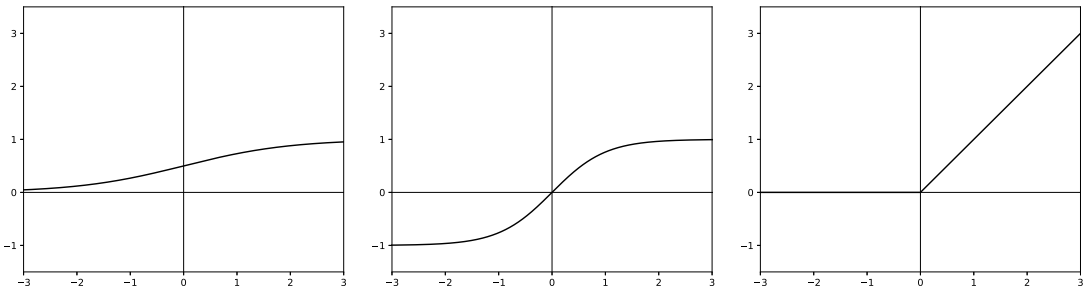
- the identity function, the layer is then called *linear* [2];
- the *logistic sigmoid function* (figure 2.2a)

$$\phi_{sigmoid}(x) = \big(1 + \exp(-x)\big)^{-1};$$

- the *hyperbolic tangent function* (figure 2.2b)

$$\phi_{tanh}(x) = 2\ \phi_{sigmoid}(2x) - 1;$$

[2] If the neural network is solely a composition of linear layers, then it can only express linear functions.



(a) Logistic sigmoid **(b)** Tangent hyperbolic **(c)** Rectified linear

Figure 2.2 – Examples of activation functions.

- the *rectified linear function* (Jarrett et al., 2009; Nair and Hinton, 2010; Glorot et al., 2011) (figure 2.2c)

$$\phi_{rect}(x) = \max(0, x).$$

These feedforward layers can typically be composed obtain the desired prediction following the recurrence

$$h^{(k+1)} = \phi_{act,(k)}(h^{(k)}W^{(k)} + b^{(k)})$$

where $h^{(0)}$ would be the input x of the artificial neural network and for $k > 0$, $h^{(k)}$ would be called a *hidden layer*, with the exception of the last, which would be the *output layer*. Each dimension of these hidden vectors are typically called *neurons*, to follow the neural network analogy, which are connected by these weights $W^{(k)}$ (see Figure 2.1). When none of the element of the weight matrix is constrained to be 0, then we call the corresponding layer *fully-connected*.

Although the default parametrization of such layer is to make the affine operation arbitrary, there is value in exploiting the particular structure in signals of interest. For instance, the local structure and limited translation equivariance of natural images and sound signals enables us to restrict this affine transformation to be a discrete convolution (LeCun and Bengio, 1994; Kavukcuoglu et al., 2010) (see also Dumoulin and Visin, 2016), resulting in improving generalization by limiting the hypothesis space, and computational efficiency, both memory-wise *parameter sharing* across the signal and computation time as convolution involves less com-

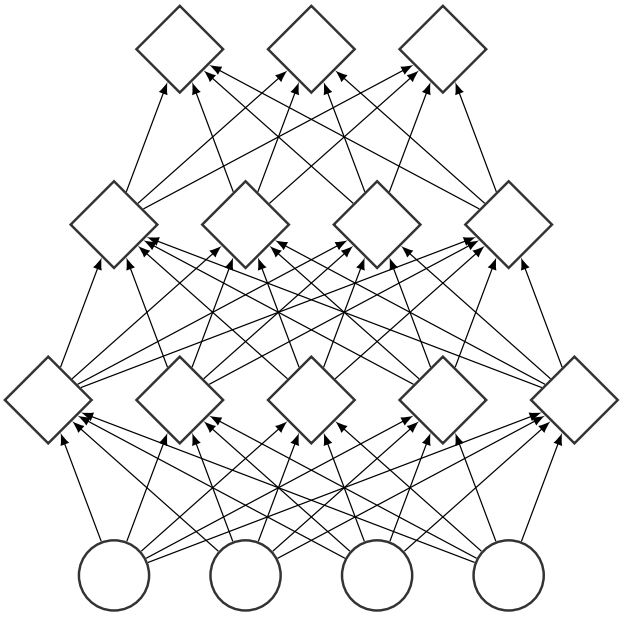


Figure 2.3 – Computational graph of a deeper feedforward neural network. The computation goes upward, the botton nodes are the inputs and the nodes above are the hidden and output units. Each hidden and output node take all the input from previous layer with different strength (defined by the weight matrix), before applying a nonlinearity if necessary.

putation (because of the underlying sparse connectivity) that can be more easily parallelized.

2.1.2 Powerful architecture

For some non-linear activation functions, even a one hidden layer neural network can approximate any continuous function with arbitrary precision on a bounded set given that this hidden layer is sufficiently large, this result is known as the *universal approximation theorem* (e.g. Cybenko, 1989; Hornik, 1991; Barron, 1993), grounding further the intuition in subsection 1.3.1 that increasing the size of the model by adding hidden units does indeed improve its capacity.

However, this capacity might come at an unbearable cost in terms of computation and generalization for shallow models, even when exploiting parallel computations. Several arguments have been made (e.g. Bengio et al., 2006; Bengio and Delalleau, 2011; Martens and Medabalimi, 2014; Montufar et al., 2014; Eldan and Shamir, 2016; Poole et al., 2016; Raghu et al., 2017; Telgarsky, 2016) demonstrating the effectiveness of favoring *deep architectures*, i.e. models with many layered computations, over shallow ones, i.e. models with fewer of these layers. The argument

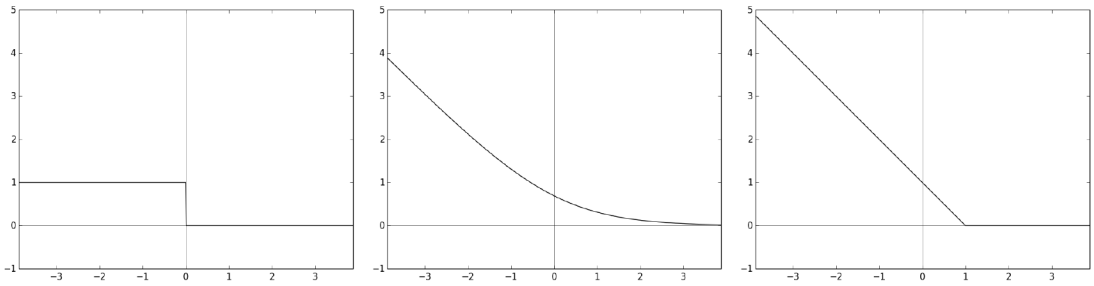
is mainly computational: a shallow architecture can require exponentially more resources (both in computation time and memory) to express the same functions that deep models can express. One of the intuitions in favor of a deep architecture has inspired by software program architecture: the more complex and abstract your program is, the more necessary it is to factor it into subroutines and subsubroutines in order to scale well.

2.2 Learning of deep models

2.2.1 Gradient-based learning

To facilitate the search in a high-dimensional parameter space Θ, deep learning models are traditionally trained with *gradient-based methods* (e.g. LeCun et al., 1998a). In order for these gradient-based methods to be used, one requires the gradient of the loss function with respect to the parameters, the steepest descent direction, to be defined almost everywhere and to be meaningful. As several loss functions of interest can be lacking these properties, e.g. the 0-1 loss, one can use a *surrogate loss function* whose minimization can hopefully result in minimizing the loss function of interest, e.g. the *logistic loss* and *hinge loss*.

In order to compute the gradient of the loss function with respect to the parameter, standard efficient implementations include generally the *backpropagation*



(a) 0-1 loss. **(b)** Logistic loss. **(c)** Hinge loss.

Figure 2.4 – Examples of loss function with respect to the matching score in the binary classification setting.

algorithm (Rumelhart et al., 1988), which combines the *chain rule* for derivatives with *dynamic programming*. If we consider the layer

$$h^{(k+1)} = \phi_{act,(k)}(h^{(k)}W^{(k)} + b^{(k)})$$

then the gradients follow the recursive equation

$$\begin{aligned}
\frac{\partial L}{\partial (W^{(k)})^T} &= \phi'_{act,(k)}(h^{(k)}W^{(k)} + b^{(k)}) \odot (h^{(k)})^T \cdot \frac{\partial L}{\partial (h^{(k+1)})^T} \\
\frac{\partial L}{\partial (b^{(k)})^T} &= \phi'_{act,(k)}(h^{(k)}W^{(k)} + b^{(k)}) \odot \frac{\partial L}{\partial (h^{(k+1)})^T} \\
\frac{\partial L}{\partial (h^{(k)})^T} &= \phi'_{act,(k)}(h^{(k)}W^{(k)} + b^{(k)}) \odot \frac{\partial L}{\partial (h^{(k+1)})^T} \cdot \left(W^{(k)}\right)^T
\end{aligned}$$

where $\odot$ is the element-wise multiplication.

While the $h^{(k)}$ quantities are computed through the forward computation, the $\frac{\partial L}{\partial (h^{(k)})^T}$ are computed during the backward computation of the derivative. As the forward computations are stored in memories, the computation of these derivatives are as efficient as the forward computation. These functionalities are implemented in frameworks including automatic differentiation toolkits like Theano (Bergstra et al., 2010; Bastien et al., 2012) and Tensorflow (Abadi et al., 2016).

2.2.2 Stochastic gradient descent

The iterative *(batch) gradient descent algorithm* (illustrated in figure 2.5) lays the foundation of most gradient-based algorithms used in deep learning. The algorithm proceeds as follow to follow a loss $\mathcal{L}_{\mathcal{D}_{\text{train}}} = \hat{\mathcal{L}}(\cdot, \mathcal{D}_{\text{train}})$: we start from a given initialization (possibly random) of the parameter $\theta_0 \in \Theta$ and at each step t we update the parameter by substracting a vector proportional to the gradient $\frac{\partial \mathcal{L}_{\mathcal{D}_{\text{train}}}}{\partial \theta_t}(\theta_t)$, scaled by a positive factor α_t called the *learning rate*. An advantage with iterative algorithms is their ability to be interrupted at any step on their way to a local minimum and still provide in theory a decent approximate solution given the computational budget. Moreover, in several successful deep learning models, even a local minimum can often reach very good performance.

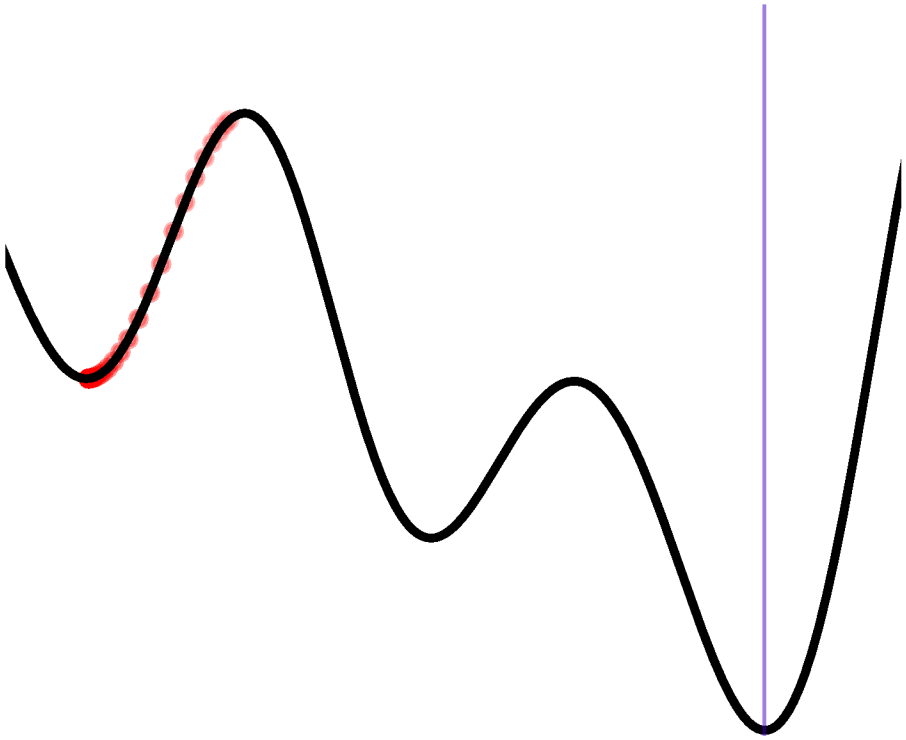


Figure 2.5 – Visualization of the gradient descent algorithm on a random function. The trajectory of gradient descent is shown in red. We can see that although the algorithm reaches a local minimum, it does not reach in this case the global minimum (represented by the purple vertical line). Whether the algorithm can reach the global minimum depends among other things on the initialization of the algorithm (Glorot et al., 2011; Sutskever et al., 2013; Sussillo and Abbott, 2014). Nonetheless, the solution obtained can reach a reasonable loss value. Best seen with colors.

Algorithm 1 Batch gradient descent algorithm

Require: Initialization of $\theta_0 \in \Theta$, learning rate sequence $(\alpha_t)_{t>=\geq 0} \in (\mathbb{R}_+^*)^{\mathbb{N}}$

$\theta \leftarrow \theta_0$

$t \leftarrow 0$

while not termination$(\ell, \theta, \mathcal{D}_{\text{train}})$ **do**

$\theta \leftarrow \theta - \alpha_t \left(\nabla_\theta \mathcal{L}_{\mathcal{D}_{\text{train}}}\right)(\theta)$

$t \leftarrow t + 1$

end while

return θ

A computational hurdle with batch gradient descent however is that computing the gradient requires computing

$$\mathcal{L}_{\mathcal{D}_{\text{train}}}(\theta_t) = \frac{1}{|\mathcal{D}_{\text{train}}|} \sum_{x \in \mathcal{D}_{\text{train}}} \ell(\theta, x),$$

which remains ultimately linear with the size of the training set, which can still be too expensive as a significant number of steps need to be taken in order to reasonably approximate a minimum. Fortunately, for a uniformly and randomly selected subset $\hat{\mathcal{M}}_{train}$ of $\mathcal{D}_{\text{train}}$, if we note

$$\mathcal{L}_{\hat{\mathcal{M}}_{train}}(\theta_t) = \frac{1}{|\hat{\mathcal{M}}_{train}|} \sum_{x \in \hat{\mathcal{M}}_{train}} \ell(\theta, x),$$

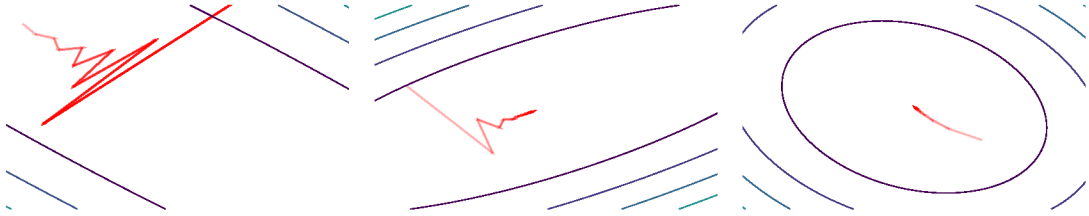
then $\frac{\partial \mathcal{L}_{\hat{\mathcal{M}}_{train}}}{\partial \theta_t}(\theta_t)$ is an unbiased stochastic approximation of $\frac{\partial \mathcal{L}_{\mathcal{D}_{\text{train}}}}{\partial \theta_t}(\theta_t)$

$$\mathbb{E}_{\hat{\mathcal{M}}_{train}}\left[\frac{\partial \mathcal{L}_{\hat{\mathcal{M}}_{train}}}{\partial \theta_t}(\theta_t)\right] = \frac{\partial \mathcal{L}_{\mathcal{D}_{\text{train}}}}{\partial \theta_t}(\theta_t)$$

whose computational cost becomes linear with the size of the subset $\hat{\mathcal{M}}_{train}$, which we call *mini-batch*. Because it uses a stochastic approximation of the gradient instead of the exact estimate, this variant is called *stochastic gradient descent* (Ermoliev, 1983) (described in Algorithm 2). Under mild assumption, this algorithm converges to a local minimum, even for a mini-batch of size 1 (Robbins and Monro, 1951; Bottou, 1998). There are of course tradeoffs to consider when picking the *(mini)batch size*, a smaller mini-batch will use less computation and memory, and the noise incurred in gradient estimation can result in regularization (Wilson and Martinez, 2003), a larger batch can also help exploit parallelizable computational processes and provide a more accurate estimation of the gradient (Goyal et al., 2017).

Algorithm 2 Stochastic gradient descent algorithm

Require: Initialization of $\theta_0 \in \Theta$, learning rate sequence $(\alpha_t)_{t>=\geq 0} \in (\mathbb{R}_+^*)^{\mathbb{N}}$
  $\theta \leftarrow \theta_0$
  $t \leftarrow 0$
  while not termination$(\ell, \theta, \mathcal{D}_{\text{train}})$ **do**
    Sample mini-batch $\hat{\mathcal{M}}_{train} \subset \mathcal{D}_{\text{train}}$ uniformly
    $\theta \leftarrow \theta - \alpha_t \left(\nabla_\theta \mathcal{L}_{\hat{\mathcal{M}}_{train}}\right)(\theta)$
    $t \leftarrow t + 1$
  end while
  return θ



(a) Gradient descent trajectory (in red) for a very poorly conditioned problem with too high learning rate. The trajectory does not converge in this case.

(b) Gradient descent trajectory (in red) for a poorly conditioned problem with appropriate learning rate. The trajectory converges in this case but fluctuates.

(c) Gradient descent trajectory (in red) for a well conditioned problem with appropriate high learning rate. The trajectory does converge well in this case.

Figure 2.6 – Examples of gradient descent trajectories on poorly conditioned and well conditioned quadratic problems, centered on the minimum.

Despite the convergence guarantees one may have with stochastic gradient descent, the speed of convergence depends on the learning rate α_t chosen: if too high, the gradient descent can overshoot in its search and stall the convergence or even diverge, if too low, the descent might be too slow to reach the minimum in a reasonable time. In high dimension, some problems, even convex, can limit the allowed range one could use for a learning rate: some directions might be well suited for larger learning rates whereas others would require smaller learning rates. These problems are then called *poorly conditioned* (Dauphin et al., 2015) (illustrated in figure 2.6).

This conditioning reflects the general geometry of the optimization problem, which is why we describe the problem in terms of *loss function landscape* (e.g Dauphin et al., 2014; Sagun et al., 2014; Choromanska et al., 2015). In terms of

pure optimization, there are several approaches to tackle this conditioning problem. *Momentum* (Nesterov et al., 2007; Sutskever et al., 2013) uses a moving average of the stochastic gradient estimate which can counter fluctuation issues for example. One can also *precondition* the problem, i.e. rearrange its geometry such that the new problem has a more appropriate conditioning, by using *second order methods* (e.g Amari, 1998; Pascanu and Bengio, 2014; Martens and Grosse, 2015) or *adaptative learning rate* (e.g Tieleman and Hinton, 2012; Dauphin et al., 2015; Wilson et al., 2017) methods. In particular, we will use *Adaptative Moment estimation* (AdaM) (Kingma and Ba, 2015) (described in Algorithm 3) for training models, despite its issue regarding convergence properties (Sashank J. Reddi, 2018).

Algorithm 3 AdaM algorithm

Require: Initialization of $\theta_0 \in \Theta$, learning rate sequence $(\alpha_t)_{t\geq 0} \in (\mathbb{R}_+^*)^{\mathbb{N}}$, decay coefficients $(\beta_1, \beta_2) \in [0, 1[$, damping coefficient $\epsilon > 0$
$\theta \leftarrow \theta_0$
$\mu_1 \leftarrow 0$
$\mu_2 \leftarrow 0$
$t \leftarrow 0$
while not termination$(\ell, \theta, \mathcal{D}_{\text{train}})$ **do**
Sample mini-batch $\mathcal{M} = \hat{\mathcal{M}}_{train} \subset \mathcal{D}_{\text{train}}$ uniformly
$\mu_1 \leftarrow \beta_1 \mu_1 + (1 - \beta_1) \left(\nabla_\theta \mathcal{L}_{\hat{\mathcal{M}}}\right)(\theta)$
$\mu_2 \leftarrow \beta_2 \mu_2 + (1 - \beta_2) \left(\left(\nabla_\theta \mathcal{L}_{\hat{\mathcal{M}}}\right)(\theta)\right)^{\odot 2}$ (gradient estimate squared element-wise)
$\theta \leftarrow \theta - \alpha_t \frac{\mu_1}{1-\beta_1^{t+1}} \sqrt{\frac{1-\beta_2^{t+1}}{\mu_2 + \epsilon}}$
$t \leftarrow t + 1$
end while
return θ

2.2.3 Gradient flow

Another view of conditioning in deep learning problem has been the issue of *gradient flow*: as gradient is estimated through backward computation, how much does the signal degrade. In particular, as the gradient is propagated backward, its norm can grow/shrink exponentially, resulting in *exploding/vanishing gradients* (Hochreiter, 1991; Bengio et al., 1994). As this issue becomes more significant with depth, this has been one of the main hurdle in training deep networks.

Several techniques have helped mitigate this difficulty, for instance new initialization techniques (e.g. Glorot et al., 2011; Sutskever et al., 2013; Sussillo and Abbott, 2014). A significant effort has been put into *architectural design* of deep models in order to improve gradient propagation.

- Glorot et al. (2011) show the advantage of using the rectified linear activation functions in deep neural networks in terms of gradient propagation, whose piecewise linear design enabled better gradient propagation than saturating nonlinearities like logistic sigmoid.

- *Batch normalization* (Ioffe and Szegedy, 2015; Desjardins et al., 2015) integrate feature normalization in the forward equation, with moments estimated from the current mini-batch, in order to alleviate gradient propagation issues.

- *Weight normalization* (Badrinarayanan et al., 2015; Salimans and Kingma, 2016; Arpit et al., 2016) reparametrizes the weights as a product of a norm and normalized direction.

- Hochreiter (1991); Bengio et al. (1994) laid the principles to design the *long-short term memory* (Hochreiter and Schmidhuber, 1997b; Zaremba et al., 2014) recurrent neural networks (Rumelhart et al., 1988), which in turn was influential in the design of *skip connections* (Lin et al., 1996), *highway networks* (Srivastava et al., 2015), and *residual networks*(He et al., 2015a, 2016). In particular residual networks replace some fully connected feedforward layers with *residual blocks* with forward equation $h^{(k+1)} = h^{(k)} + f_{res,(k)}(h^{(k)})$ where $f_{res,(k)}$ is a small and shallow neural network (see figure 2.7). These *residual connections* possibly solve a *shattered gradient* (Balduzzi et al., 2017) problem, i.e. when the gradients with respect to the parameters are similar to white noise.

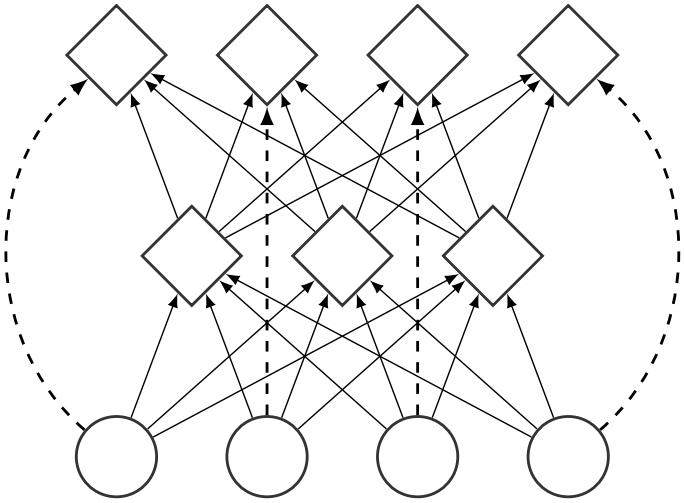


Figure 2.7 – Computational graph of a typical residual block. The output of the small neural network is added to the input. This operation represented by dashed arrows which are called the *residual connections*.

There is arguably a relationship between architectural design and adaptative gradient methods through *reparametrization*. If $\eta = \Phi^{-1}(\theta)$ a reparametrization of θ (with Φ bijective), then by taking the new loss function $\mathcal{L}^{(\eta)}_{\mathcal{D}_{\text{train}}} = \mathcal{L}_{\mathcal{D}_{\text{train}}} \circ \Phi$, the gradient in this reparametrized problem has the form

$$\left(\nabla_\eta \mathcal{L}^{(\eta)}_{\mathcal{D}_{\text{train}}}\right)(\eta) = \left(\nabla_\theta \mathcal{L}_{\mathcal{D}_{\text{train}}}\right)(\theta) \cdot \left(\nabla_\eta \Phi\right)(\eta),$$

resulting in a different gradient descent algorithm (reminiscent of the *mirror descent algorithm*, Beck and Teboulle, 2003) by manipulation of the problem geometry

$$\begin{aligned}
\theta_{t+1} &= \Phi(\eta_{t+1}) \\
&= \Phi\left(\eta_t - \alpha_t \left(\nabla_\eta \mathcal{L}^{(\eta)}_{\mathcal{D}_{\text{train}}}\right)(\eta_t)\right) \\
&= \Phi\left(\Phi^{-1}(\theta_t) - \alpha_t \left(\nabla_\theta \mathcal{L}_{\mathcal{D}_{\text{train}}}\right)(\theta_t) \cdot \left(\nabla_\eta \Phi\right)\left(\Phi^{-1}(\theta_t)\right)\right)
\end{aligned}$$

as shown by Sohl-Dickstein (2012) when Φ is linear.

2.3 Generalizability of deep learning

Although expressivity and fast computation enabled successful applications of deep learning models to *object recognition in images* (e.g Krizhevsky et al., 2012; Goodfellow et al., 2014a; Simonyan and Zisserman, 2015; Szegedy et al., 2015; He et al., 2016, 2017), *machine translation* (e.g. Cho et al., 2014; Sutskever et al., 2014; Bahdanau et al., 2015; Wu et al., 2016; Gehring et al., 2016), and *speech recognition* (e.g. Dahl et al., 2012; Hinton et al., 2012; Graves et al., 2013; Hannun et al., 2014; Chorowski et al., 2015; Chan et al., 2016; Collobert et al., 2016), it does not solely explain their success. In order for learning to succeed, the algorithms need to have some degree of generalizability.

A recent study (Zhang et al., 2017) highlighted the well-known issue that statistical learning theory notions like *Vapnik–Chervonenkis dimension* (Vapnik and Chervonenkis, 2015) and *Rademacher complexity* (Bartlett and Mendelson, 2002) used to obtain *probably approximately correct (PAC)* generalization bounds cannot be applied to deep learning models to obtain useful bounds. Indeed, the only guarantee they provide is a generalization error lower than 100%, they are therefore called *vacuous* (Dziugaite and Roy, 2017).

One of the promising approaches to explaining generalization in deep learning is the holistic approach of *stochastic approximation* (Nesterov and Vial, 2008; Bottou et al., 2016), encompassing the joint issues of approximation, computation, and stochasticity. In particular, the notion of *uniform stability* (Bousquet and Elisseeff, 2002) allowed to explore an interesting path in building generalization bounds in restricted cases (Hardt et al., 2016; Gonen and Shalev-Shwartz, 2017). Unfortunately, this approach does not account for the particularity of the problem at hand whereas, as stated earlier, its complexity should modulate the tradeoff between approximation quality, computation, and estimation reliability.

The *PAC-Bayes* framework (McAllester, 1999; Langford and Caruana, 2001; Dziugaite and Roy, 2017; Neyshabur et al., 2017) provides nonvacuous data-dependent bounds on the generalization gap. It relies on defining an *a priori distribution* p (i.e. defined before any training) on the parameter θ and a distribution q defined by training, and gives an expression of the generalization gap as function of the

Kullback-Leibler divergence (Kullback and Leibler, 1951) between p and q

$$KL(q\|p) = \mathbb{E}_{\theta\sim q}\left[\log\left(\frac{q(\theta)}{p(\theta)}\right)\right],$$

grounding the intuition put forward in Hinton and Van Camp (1993) of using this quantity as a regularization term, an approach known as *minimum description length* (MDL).

Heskes and Kappen (1993) formulate the limit sampling process of stochastic gradient descent in the continuous time limit, on which Mandt et al. (2016); Smith and Le (2017) build on in an attempt to bridge between stochastic gradient descent and *variational inference* (Neal and Hinton, 1998; Wainwright et al., 2008). This contrasts slightly with works (Welling and Teh, 2011; Ahn et al., 2012) that relate stochastic gradient descent with *Markov Chain Monte Carlo* (Andrieu et al., 2003) approaches to inference through *Hamiltonian Monte Carlo* (Neal et al., 2011).

Dziugaite and Roy (2017); Neyshabur et al. (2017) attempt to relate the intuition of this PAC-Bayes approach to the conjecture formulated in Hochreiter and Schmidhuber (1997a) that *flat minima*, i.e. around which the error is approximately constant, generalize better than *sharp* ones, i.e. around which the error varies largely. This conjecture has been lasting since and was maintained by Chaudhari et al. (2017); Keskar et al. (2017); Smith and Le (2017); Krueger et al. (2017).

3 On the relevance of loss function geometry for generalization

While several works argue in favor of the conjecture that flat minima generalize better than sharp ones, the definition of said flatness/sharpness varies from one manuscript to the other. However, the intuition behind that notion is fairly simple: if one imagines the error as a one-dimensional curve, a minimum is flat if there is a wide region around it with roughly the same error, otherwise the minimum is sharp. When moving to higher dimensional spaces, defining flatness becomes more complicated. In Hochreiter and Schmidhuber (1997a) it is defined as the size of the connected region around the minimum where the training loss is relatively similar. Chaudhari et al. (2017) relies, in contrast, on the curvature of the second order structure around the minimum, while Keskar et al. (2017) looks at the maximum loss in a bounded neighbourhood of the minimum. All these works rely on the fact that flatness results in robustness to low precision arithmetic or noise in the parameter space, which, using a minimum description length-based argument, suggests a low expected overfitting (Bishop, 1993).

We will demonstrate how several common architectures and parametrizations in deep learning are already at odds with this conjecture, requiring at least some degree of refinement in its statements. In particular, we show how the geometry of the associated parameter space can alter the ranking between prediction functions when considering several measures of flatness/sharpness. We believe the reason for this contradiction stems from the Bayesian arguments about KL-divergence made to justify the generalization ability of flat minima (Hinton and Van Camp, 1993; Dziugaite and Roy, 2017). Indeed, *the Kullback-Liebler divergence is invariant to reparametrization whereas the notion of "flatness" is not.* The demonstrations of Hochreiter and Schmidhuber (1997a) are approximately based on a Gibbs formalism and rely on strong assumptions and approximations that can compromise the applicability of the argument, including the assumption of a discrete function space.

Work (Dinh et al., 2017) done in collaboration with Dr. Razvan Pascanu, Dr. Samy Bengio, and Pr. Yoshua Bengio.

3.1 Definitions of flatness/sharpness

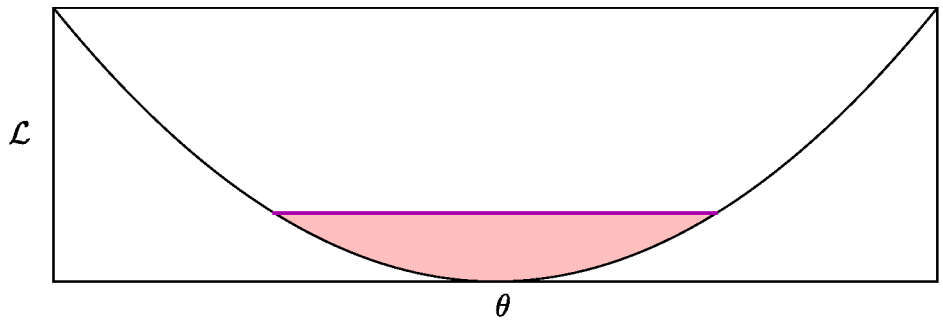


Figure 3.1 – An illustration of the notion of flatness. The loss $\mathcal{L}$ as a function of θ is plotted in black. If the height of the red area is ϵ, the width will represent the volume ϵ-flatness. If the width is 2ϵ, the height will then represent the ϵ-sharpness. Best seen with colors.

For conciseness, we will restrict ourselves to supervised scalar output problems, but several conclusions in this chapter can apply to other problems as well. We will consider a function f that takes as input an element x from an input space $\mathcal{X}$ and outputs a scalar y. We will denote by f_θ the prediction function. This prediction function will be parametrized by a parameter vector θ in a parameter space Θ. Often, this prediction function will be over-parametrized and two parameters $(\theta, \theta') \in \Theta^2$ that yield the same prediction function everywhere, $\forall x \in \mathcal{X}, f_\theta(x) = f_{\theta'}(x)$, are called *observationally equivalent*. The model aims to minimize a continuous loss function $\mathcal{L}$ (since our reasoning in this chapter applies to both training and validation, we will drop the subscript) which takes as argument the prediction function f_θ. We will often think of the loss $\mathcal{L}$ as a function of θ and adopt the notation $\mathcal{L}(\theta)$.

The notion of flatness/sharpness of a minimum is relative, therefore we will discuss metrics that can be used to compare the relative flatness between two minima. In this section we will formalize three used definitions of flatness in the literature.

Hochreiter and Schmidhuber (1997a) defines a flat minimum as "a large connected region in weight space where the error remains approximately constant". We interpret this formulation as follows:

Definition 1. *Given $\epsilon > 0$, a minimum θ, and a loss $\mathcal{L}$, we define $C(\mathcal{L}, \theta, \epsilon)$ as the largest (using inclusion as the partial order over the subsets of Θ) connected*

set containing θ such that $\forall \theta' \in C(\mathcal{L}, \theta, \epsilon), \mathcal{L}(\theta') < \mathcal{L}(\theta) + \epsilon$. The ϵ-flatness will be defined as the volume of $C(\mathcal{L}, \theta, \epsilon)$. We will call this measure the volume ϵ-flatness.

In Figure 3.1, $C(\mathcal{L}, \theta, \epsilon)$ will be the purple line at the top of the red area if the height is ϵ and its volume will simply be the length of the purple line.

Flatness can also be defined using the local curvature of the loss function around the minimum if it is a critical point [1]. Chaudhari et al. (2017); Keskar et al. (2017) suggest that this information is encoded in the eigenvalues of the Hessian. However, in order to compare how flat one minimum is versus another, the eigenvalues need to be reduced to a single number. Here we consider the *spectral norm and trace of the Hessian*, two typical measurements of the eigenvalues of a matrix.

Additionally Keskar et al. (2017) define the notion of ϵ-sharpness. In order to make proofs more readable, we will slightly modify their definition. However, because of norm equivalence in finite dimensional space, our results will transfer to the original definition in full space as well. Our modified definition is the following:

Definition 2. *Let $B_2(\epsilon, \theta)$ be an Euclidean ball centered on a minimum θ with radius ϵ. Then, for a non-negative valued loss function $\mathcal{L}$, the ϵ-sharpness will be defined as proportional to*

$$\max_{\theta' \in B_2(\epsilon, \theta)} \left(\frac{\mathcal{L}(\theta') - \mathcal{L}(\theta)}{1 + \mathcal{L}(\theta)} \right).$$

In Figure 3.1, if the width of the red area is 2ϵ then the height of the red area is $\max_{\theta' \in B_2(\epsilon, \theta)} (\mathcal{L}(\theta') - \mathcal{L}(\theta)) \geq 0$.

ϵ-sharpness can be related to the spectral norm $\left\| (\nabla^2 \mathcal{L})(\theta) \right\|_2$ of the Hessian $(\nabla^2 \mathcal{L})(\theta)$. Indeed, a second-order Taylor expansion of $\mathcal{L}$ around a critical point minimum is written

$$\begin{aligned} \mathcal{L}(\theta') = \mathcal{L}(\theta) + \frac{1}{2} (\theta' - \theta) (\nabla^2 \mathcal{L})(\theta)(\theta' - \theta)^T \\ + o(\|\theta' - \theta\|_2^2). \end{aligned}$$

[1] In this chapter, we will often assume that is the case when dealing with Hessian-based measures in order to have them well-defined.

In this second order approximation, the ϵ-sharpness at θ would be

$$\frac{\|\|(\nabla^2\mathcal{L})(\theta)\|\|_2\epsilon^2}{2\left(1+\mathcal{L}(\theta)\right)}.$$

3.2 Properties of deep rectified Networks

Before moving forward to our results, in this section we first introduce the notation used in the rest of chapter. Most of our results, for clarity, will be on the deep rectified feedforward networks with a linear output layer that we describe below, though they can easily be extended to other architectures (e.g. convolutional, etc.).

Definition 3. *Given K weight matrices $(\theta_k)_{k\leq K}$ with $n_k = dim\left(vec(\theta_k)\right)$ and $n = \sum_{k=1}^{K} n_k$, the output y of a deep rectified feedforward networks with a linear output layer is:*

$$y = \phi_{rect}\left(\phi_{rect}\left(\cdots\phi_{rect}(x\cdot\theta_1)\cdots\right)\cdot\theta_{K-1}\right)\cdot\theta_K,$$

where

- *x is the input to the model, a high-dimensional vector*
- *vec reshapes a matrix into a vector.*

Note that in our definition we excluded the bias terms, usually found in any neural architecture. This is done mainly for convenience, to simplify the rendition of our arguments. However, the arguments can be extended to the case that includes biases. Another choice is that of the linear output layer. Having an output activation function does not affect our argument either: since the loss is a function of the output activation, it can be rephrased as a function of linear pre-activation.

Deep rectifier models have certain properties that allows us in section 3.4 to arbitrary manipulate the flatness of a minimum.

An important topic for optimization of neural networks is understanding the non-Euclidean geometry of the parameter space as imposed by the neural architecture (see, for example, Amari, 1998). In principle, when we take a step in parameter space what we expect to control is the change in the behavior of the model (i.e.

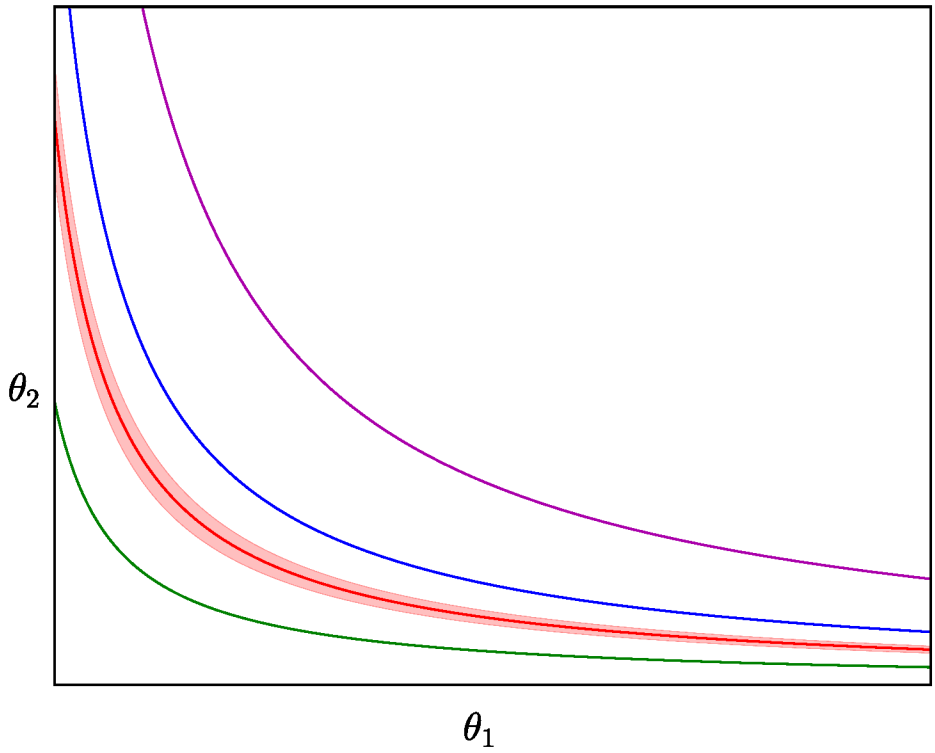


Figure 3.2 – An illustration of the effects of non-negative homogeneity. The graph depicts level curves of the behavior of the loss L embedded into the two dimensional parameter space with the axis given by θ_1 and θ_2. Specifically, each line of a given color corresponds to the parameter assignments (θ_1, θ_2) that result observationally in the same prediction function f_θ. Best seen with colors.

the mapping of the input x to the output y). In principle we are not interested in the parameters per se, but rather only in the mapping they represent.

If one defines a measure for the change in the behavior of the model, which can be done under some assumptions, then, it can be used to define, at any point in the parameter space, a metric that says what is the equivalent change in the parameters for a unit of change in the behavior of the model. As it turns out, for neural networks, this metric is not constant over Θ. Intuitively, the metric is related to the curvature, and since neural networks can be highly non-linear, the curvature will not be constant. See Amari (1998); Pascanu and Bengio (2014) for more details. Coming back to the concept of flatness or sharpness of a minimum, this metric should define the flatness.

However, the geometry of the parameter space is more complicated. Regardless

of the measure chosen to compare two instantiations of a neural network, because of the structure of the model, it also exhibits a large number of symmetric configurations that result in exactly the same behavior. Because the rectifier activation has the non-negative homogeneity property, as we will see shortly, one can construct a continuum of points that lead to the same behavior, hence the metric is singular. It means that one can exploit these directions in which the model stays unchanged to shape the neighbourhood around a minimum in such a way that, by most definitions of flatness, this property can be controlled. See Figure 3.2 for a visual depiction, where the flatness (given here as the distance between the different level curves) can be changed by moving along the curve.

Let us redefine, for convenience, the *non-negative homogeneity* property (Neyshabur et al., 2015; Lafond et al., 2016) below. Note that beside this property, the reason for studying the rectified linear activation is for its widespread adoption (Krizhevsky et al., 2012; Simonyan and Zisserman, 2015; Szegedy et al., 2015; He et al., 2016).

Definition 4. *A given a function ϕ is* non-negative homogeneous *if*

$$\forall(z, \alpha) \in \mathbb{R} \times \mathbb{R}^{+}, \phi(\alpha z) = \alpha\phi(z)$$

.

Theorem 1. *The rectified function $\phi_{rect}(x) = \max(x, 0)$ is non-negative homogeneous.*

Proof. Follows trivially from the constraint that $\alpha > 0$, given that $x > 0 \Rightarrow \alpha x > 0$, iff $\alpha > 0$.

□

For a deep rectified neural network without biases it means that:

$$\phi_{rect}\big(x \cdot (\alpha\theta_1)\big) \cdot \theta_2 = \phi_{rect}(x \cdot \theta_1) \cdot (\alpha\theta_2),$$

meaning that for this one (hidden) layer neural network, the parameters $(\alpha\theta_1, \theta_2)$ is observationally equivalent to $(\theta_1, \alpha\theta_2)$. This observational equivalence similarly holds for convolutional layers.

Given this non-negative homogeneity, if $(\theta_1, \theta_2) \neq (0, 0)$ then $\big\{(\alpha\theta_1, \alpha^{-1}\theta_2), \alpha > 0\big\}$ is an infinite set of observationally equivalent parameters, inducing a strong

non-identifiability in this learning scenario. Other models like *deep linear networks* (Saxe et al., 2013), *leaky rectifiers* (He et al., 2015b) or *maxout networks* (Goodfellow et al., 2013) also have this non-negative homogeneity property.

In what follows we will rely on such transformations, in particular we will rely on the following definition:

Definition 5. *For a single hidden layer rectifier feedforward network we define the family of transformations*

$$T_\alpha : (\theta_1, \theta_2) \mapsto (\alpha\theta_1, \alpha^{-1}\theta_2)$$

which we refer to as an α-scale transformation.

Note that an α-scale transformation will not affect the generalization, as the behavior of the function is identical. Also while the transformation is only defined for a single layer rectified feedforward network, it can trivially be extended to any architecture including a rectified network as a submodule, e.g. a deep rectified feedforward network. For simplicity and readability we will rely on this definition.

3.3 Rectified neural networks and Lipschitz continuity

Relative to recent works (Hardt et al., 2016; Gonen and Shalev-Shwartz, 2017) assuming *Lipschitz continuity* of the loss function to derive uniform stability bound, we make the following observation:

Theorem 2. *For a one-hidden layer rectified neural network of the form*

$$y = \phi_{rect}(x \cdot \theta_1) \cdot \theta_2,$$

if $\mathcal{L}$ is not constant, then it is not Lipschitz continuous.

Proof. Since a Lipschitz function is necessarily absolutely continuous, we will consider the cases where $\mathcal{L}$ is absolutely continuous. First, if L has zero gradient almost everywhere, then $\mathcal{L}$ is constant.

Now, if there is a point θ with non-zero gradient, then by writing

$$(\nabla\mathcal{L})(\theta_1,\theta_2) = [(\nabla_{\theta_1}\mathcal{L})(\theta_1,\theta_2) \\ (\nabla_{\theta_2}\mathcal{L})(\theta_1,\theta_2)],$$

we have

$$(\nabla\mathcal{L})(\alpha\theta_1,\alpha^{-1}\theta_2) = [\alpha^{-1}(\nabla_{\theta_1}\mathcal{L})(\theta_1,\theta_2) \\ \alpha(\nabla_{\theta_2}\mathcal{L})(\theta_1,\theta_2)].$$

Without loss of generality, we consider $(\nabla_{\theta_1}\mathcal{L})(\theta_1,\theta_2) \neq 0$. Then the limit of the norm

$$\|(\nabla\mathcal{L})(\alpha\theta_1,\alpha^{-1}\theta_2)\|_2^2 = \alpha^{-2}\|(\nabla_{\theta_1}\mathcal{L})(\theta_1,\theta_2)\|_2^2 \\ + \alpha^2\|(\nabla_{\theta_2}\mathcal{L})(\theta_1,\theta_2)\|_2^2$$

of the gradient goes to $+\infty$ as α goes to 0. Therefore, $\mathcal{L}$ is not Lipschitz continuous. □

This result can be generalized to several other models containing a one-hidden layer rectified neural network, including deeper rectified networks.

3.4 Deep rectified networks and flat minima

In this section we exploit the resulting strong non-identifiability to showcase a few shortcomings of some definitions of flatness. Although α-scale transformation does not affect the function represented, it allows us to significantly decrease several measures of flatness. For another definition of flatness, the α-scale transformation shows that all minima are equally flat.

3.4.1 Volume ϵ-flatness

Theorem 3. *For a one-hidden layer rectified neural network of the form*

$$y = \phi_{rect}(x \cdot \theta_1) \cdot \theta_2,$$

and a minimum $\theta = (\theta_1, \theta_2)$, such that $\theta_1 \neq 0$ and $\theta_2 \neq 0$, $\forall \epsilon > 0$ $C(\mathcal{L}, \theta, \epsilon)$ has an infinite volume.

We will not consider the solution θ where any of the weight matrices θ_1, θ_2 is zero, $\theta_1 = 0$ or $\theta_2 = 0$, as it results in a constant function which we will assume to give poor training performance. For $\alpha > 0$, the α-scale transformation $T_\alpha : (\theta_1, \theta_2) \mapsto (\alpha\theta_1, \alpha^{-1}\theta_2)$ has Jacobian determinant $\alpha^{n_1 - n_2}$, where once again $n_1 = dim\,(\text{vec}(\theta_1))$ and $n_2 = dim\,(\text{vec}(\theta_2))$. Note that the Jacobian determinant of this linear transformation is the change of volume induced by T_α and $T_\alpha \circ T_\beta = T_{\alpha\beta}$. We show below that there is a connected region containing θ with infinite volume and where the error remains approximately constant.

Proof. We will first introduce a small region with approximately constant error around θ with non-zero volume. Given $\epsilon > 0$ and if we consider the loss function continuous with respect to the parameter, $C(\mathcal{L}, \theta, \epsilon)$ is an open set containing θ. Since we also have $\theta_1 \neq 0$ and $\theta_2 \neq 0$, let $r > 0$ such that the L_∞ ball $B_\infty(r, \theta)$ is in $C(\mathcal{L}, \theta, \epsilon)$ and has empty intersection with $\{\theta', \theta'_1 = 0\}$. Let $v = (2r)^{n_1 + n_2} > 0$ the volume of $B_\infty(r, \theta)$.

Since the Jacobian determinant of T_α is the multiplicative change of induced by T_α, the volume of $T_\alpha\left(B_\infty(r, \theta)\right)$ is $v\alpha^{n_1 - n_2}$. If $n_1 \neq n_2$, we can arbitrarily grow the volume of $T_\alpha\left(B_\infty(r, \theta)\right)$, with error within an ϵ-interval of $\mathcal{L}(\theta)$, by having α tends to $+\infty$ if $n_1 > n_2$ or to 0 otherwise.

If $n_1 = n_2$, $\forall \alpha' > 0, T_{\alpha'}\left(B_\infty(r, \theta)\right)$ has volume v. Let $C' = \bigcup_{\alpha' > 0} T_{\alpha'}\left(B_\infty(r, \theta)\right)$. C' is a connected region where the error remains approximately constant, i.e. within an ϵ-interval of $\mathcal{L}(\theta)$.

Let $\alpha = 2\frac{\|\theta_1\|_\infty + r}{\|\theta_1\|_\infty - r}$. Since

$$B_\infty(r, \theta) = B_\infty(r, \theta_1) \times B_\infty(r, \theta_2),$$

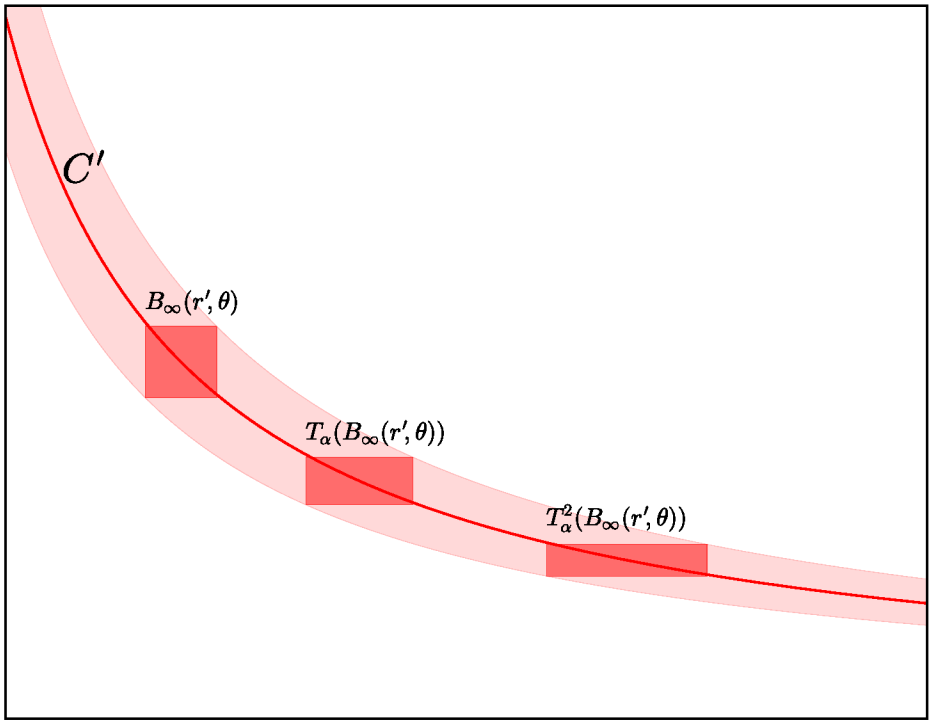


Figure 3.3 – An illustration of how we build different disjoint volumes using T_α. In this two-dimensional example, $T_\alpha\left(B_\infty(r', \theta)\right)$ and $B_\infty(r', \theta)$ have the same volume. $B_\infty(r', \theta), T_\alpha\left(B_\infty(r', \theta)\right), T_\alpha^2\left(B_\infty(r', \theta)\right), \ldots$ will therefore be a sequence of disjoint constant volumes. C' will therefore have an infinite volume. Best seen with colors.

where $\times$ is the Cartesian set product, we have

$$T_\alpha\left(B_\infty(r, \theta)\right) = B_\infty(\alpha r, \alpha\theta_1) \times B_\infty(\alpha^{-1} r, \alpha^{-1}\theta_2).$$

Therefore, $T_\alpha\left(B_\infty(r, \theta)\right) \cap B_\infty(r, \theta) = \emptyset$ (see Figure 3.3).

Similarly, $B_\infty(r, \theta), T_\alpha\left(B_\infty(r, \theta)\right), T_\alpha^2\left(B_\infty(r, \theta)\right), \ldots$ are disjoint and have volume v. We have also $T_\alpha^k\left(B_\infty(r', \theta)\right) = T_{\alpha^k}\left(B_\infty(r', \theta)\right) \in C'$. The volume of C' is then lower bounded by $0 < v + v + v + \cdots$ and is therefore infinite. $C(\mathcal{L}, \theta, \epsilon)$ has then infinite volume too, making the volume ϵ-flatness of θ infinite. □

This theorem can generalize to rectified neural networks in general with a similar proof. Given that every minimum has an infinitely large region (volume-wise) in which the error remains approximately constant, that means that every minimum would be infinitely flat according to the volume ϵ-flatness. Since all minima are equally flat, it is not possible to use volume ϵ-flatness to gauge the generalization

property of a minimum.

3.4.2 Hessian-based measures

The non-Euclidean geometry of the parameter space, coupled with the manifolds of observationally equal behavior of the model, allows one to move from one region of the parameter space to another, changing the curvature of the model without actually changing the function. This approach has been used with success to improve optimization, by moving from a region of high curvature to a region of well behaved curvature (e.g. Desjardins et al., 2015; Salimans and Kingma, 2016). In this section we look at two widely used measures of the Hessian, the spectral radius and trace, showing that either of these values can be manipulated without actually changing the behavior of the function. If the flatness of a minimum is defined by any of these quantities, then it could also be easily manipulated.

Theorem 4. *The gradient and Hessian of the loss $\mathcal{L}$ with respect to θ can be modified by T_α.*

Proof.

$$\mathcal{L}(\theta_1, \theta_2) = \mathcal{L}(\alpha\theta_1, \alpha^{-1}\theta_2),$$

[2] we have then by differentiation

$$(\nabla\mathcal{L})(\theta_1, \theta_2) = (\nabla\mathcal{L})(\alpha\theta_1, \alpha^{-1}\theta_2) \begin{bmatrix} \alpha\mathbb{I}_{n_1} & 0 \\ 0 & \alpha^{-1}\mathbb{I}_{n_2} \end{bmatrix}$$
$$\Leftrightarrow (\nabla\mathcal{L})(\alpha\theta_1, \alpha^{-1}\theta_2) = (\nabla\mathcal{L})(\theta_1, \theta_2) \begin{bmatrix} \alpha^{-1}\mathbb{I}_{n_1} & 0 \\ 0 & \alpha\mathbb{I}_{n_2} \end{bmatrix}$$

and

$$(\nabla^2\mathcal{L})(\alpha\theta_1, \alpha^{-1}\theta_2)$$
$$= \begin{bmatrix} \alpha^{-1}\mathbb{I}_{n_1} & 0 \\ 0 & \alpha\mathbb{I}_{n_2} \end{bmatrix} (\nabla^2\mathcal{L})(\theta_1, \theta_2) \begin{bmatrix} \alpha^{-1}\mathbb{I}_{n_1} & 0 \\ 0 & \alpha\mathbb{I}_{n_2} \end{bmatrix}.$$

□

[2] Note that this equation also proves that the Fisher information matrix is singular.

Sharpest direction Through these transformations we can easily find, for any critical point which is a minimum with non-zero Hessian, an observationally equivalent parameter whose Hessian has an arbitrarily large spectral norm.

Theorem 5. *For a one-hidden layer rectified neural network of the form*

$$y = \phi_{rect}(x \cdot \theta_1) \cdot \theta_2,$$

and critical point $\theta = (\theta_1, \theta_2)$ *being a minimum for* $\mathcal{L}$*, such that* $(\nabla^2\mathcal{L})(\theta) \neq 0$*,* $\forall M > 0, \exists \alpha > 0, \left|\left|\left|\left(\nabla^2\mathcal{L}\right)\left(T_\alpha(\theta)\right)\right|\right|\right|_2 \geq M$ *where* $\left|\left|\left|\left(\nabla^2\mathcal{L}\right)\left(T_\alpha(\theta)\right)\right|\right|\right|_2$ *is the spectral norm of* $\left(\nabla^2\mathcal{L}\right)\left(T_\alpha(\theta)\right)$*.*

Proof. The trace of a symmetric matrix is the sum of its eigenvalues and a real symmetric matrix can be diagonalized in $\mathbb{R}$, therefore if the Hessian is non-zero, there is at least one non-zero positive diagonal element. Without loss of generality, we will assume that this non-zero element of value $\gamma > 0$ corresponds to an element in θ_1. Therefore the Frobenius norm $\left|\left|\left|\left(\nabla^2\mathcal{L}\right)\left(T_\alpha(\theta)\right)\right|\right|\right|_F$ of

$$\begin{aligned}
&(\nabla^2\mathcal{L})(\alpha\theta_1, \alpha^{-1}\theta_2) \\
&= \begin{bmatrix} \alpha^{-1}\mathbb{I}_{n_1} & 0 \\ 0 & \alpha\mathbb{I}_{n_2} \end{bmatrix} (\nabla^2\mathcal{L})(\theta_1, \theta_2) \begin{bmatrix} \alpha^{-1}\mathbb{I}_{n_1} & 0 \\ 0 & \alpha\mathbb{I}_{n_2} \end{bmatrix}.
\end{aligned}$$

is lower bounded by $\alpha^{-2}\gamma$.

Since all norms are equivalent in finite dimension, there exists a constant $r > 0$ such that $r\left|\left|\left|A\right|\right|\right|_F \leq \left|\left|\left|A\right|\right|\right|_2$ for all symmetric matrices A. So by picking $\alpha < \sqrt{\frac{r\gamma}{M}}$, we are guaranteed that $\left|\left|\left|\left(\nabla^2\mathcal{L}\right)\left(T_\alpha(\theta)\right)\right|\right|\right|_2 \geq M$. □

Any minimum with non-zero Hessian will be observationally equivalent to a minimum whose Hessian has an arbitrarily large spectral norm. Therefore for any minimum in the loss function, if there exists another minimum that generalizes better then there exists another minimum that generalizes better and is also sharper according the spectral norm of the Hessian. The spectral norm of critical points' Hessian becomes as a result less relevant as a measure of potential generalization error. Moreover, since the spectral norm lower bounds the trace for a positive semi-definite symmetric matrix, the same conclusion can be drawn for the trace.

Many directions However, some notion of sharpness might take into account the entire eigenspectrum of the Hessian as opposed to its largest eigenvalue, for instance, Chaudhari et al. (2017) describe the notion of *wide valleys*, allowing the presence of very few large eigenvalues. We can generalize the transformations between observationally equivalent parameters to deeper neural networks with $K-1$ hidden layers: for $\alpha_k > 0, T_\alpha : (\theta_k)_{k\leq K} \mapsto (\alpha_k\theta_k)_{k\in K}$ with $\prod_{k=1}^{K} \alpha_k = 1$. If we define

$$D_\alpha = \begin{bmatrix} \alpha_1^{-1}\mathbb{I}_{n_1} & 0 & \cdots & 0 \\ 0 & \alpha_2^{-1}\mathbb{I}_{n_2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \alpha_K^{-1}\mathbb{I}_{n_K} \end{bmatrix}$$

then the first and second derivatives at $T_\alpha(\theta)$ will be

$$\begin{aligned} (\nabla\mathcal{L})\left(T_\alpha(\theta)\right) &= (\nabla\mathcal{L})(\theta)D_\alpha \\ (\nabla^2\mathcal{L})\left(T_\alpha(\theta)\right) &= D_\alpha(\nabla^2\mathcal{L})(\theta)D_\alpha. \end{aligned}$$

We will show to which extent you can increase several eigenvalues of $(\nabla^2\mathcal{L})\left(T_\alpha(\theta)\right)$ by varying α.

Definition 6. *For each $n\times n$ matrix A, we define the vector $\lambda(A)$ of sorted singular values of A with their multiplicity $\lambda_1(A) \geq \lambda_2(A) \geq \cdots \geq \lambda_n(A)$.*

If A is symmetric positive semi-definite, $\lambda(A)$ is also the vector of its sorted eigenvalues.

Theorem 6. *For a $(K-1)$-hidden layer rectified neural network of the form*

$$y = \phi_{rect}(\phi_{rect}(\cdots\phi_{rect}(x\cdot\theta_1)\cdots)\cdot\theta_{K-1})\cdot\theta_K,$$

and critical point $\theta = (\theta_k)_{k\leq K}$ being a minimum for L, such that $(\nabla^2\mathcal{L})(\theta)$ has rank $r = rank((\nabla^2\mathcal{L})(\theta))$, $\forall M > 0, \exists\alpha > 0$ such that $(r - \min_{k\leq K}(n_k))$ eigenvalues are greater than M.

Proof. For simplicity, we will note $\sqrt{M}$ the *principal square root* of a symmetric positive-semidefinite matrix M. The eigenvalues of $\sqrt{M}$ are the square root of the eigenvalues of M and are its *singular values*. By definition, the *singular values* of

$\sqrt{(\nabla^2\mathcal{L})(\theta)}D_\alpha$ are the square root of the eigenvalues of $D_\alpha(\nabla^2\mathcal{L})(\theta)D_\alpha$. Without loss of generality, we consider $\min_{k\leq K}(n_k) = n_K$ and choose $\forall k < K, \alpha_k = \beta^{-1}$ and $\alpha_K = \beta^{K-1}$. Since D_α and $\sqrt{(\nabla^2\mathcal{L})(\theta)}$ are positive symmetric semi-definite matrices, we can apply the multiplicative Horn inequalities (Klyachko, 2000) on singular values of the product $\sqrt{(\nabla^2\mathcal{L})(\theta)}D_\alpha$:

$$\begin{aligned}&\forall i \leq n, j \leq (n - n_K),\\ &\qquad \lambda_{i+j-n}\left((\nabla^2\mathcal{L})(\theta)D_\alpha^2\right) \geq \lambda_i\left((\nabla^2\mathcal{L})(\theta)\right)\beta^2.\end{aligned}$$

By choosing $\beta > \sqrt{\frac{M}{\lambda_r((\nabla^2\mathcal{L})(\theta))}}$, since we have $\forall i \leq r, \lambda_i\left((\nabla^2\mathcal{L})(\theta)\right) \geq \lambda_r\left((\nabla^2\mathcal{L})(\theta)\right) > 0$ we can conclude that

$$\begin{aligned}&\forall i \leq (r - n_K),\\ \lambda_i\left((\nabla^2\mathcal{L})\left(T_\alpha(\theta)\right)\right) &= \lambda_i\left((\nabla^2\mathcal{L})(\theta)D_\alpha^2\right)\\ &\geq \lambda_{i+n_k}\left((\nabla^2\mathcal{L})(\theta)\right)\beta^2\\ &\geq \lambda_r\left((\nabla^2\mathcal{L})(\theta)\right)\beta^2 > M.\end{aligned}$$

□

It means that there exists an observationally equivalent parameter with at least $(r - \min_{k\leq K}(n_k))$ arbitrarily large eigenvalues. Since Sagun et al. (2016) seems to suggests that rank deficiency in the Hessian is due to over-parametrization of the model, one could conjecture that $(r - \min_{k\leq K}(n_k))$ can be high for thin and deep neural networks, resulting in a majority of large eigenvalues. Therefore, it would still be possible to obtain an equivalent parameter with large Hessian eigenvalues, i.e. sharp in multiple directions.

3.4.3 ϵ-sharpness

Full space We have redefined for $\epsilon > 0$ the ϵ-sharpness of Keskar et al. (2017) as follow

$$\max_{\theta' \in B_2(\epsilon,\theta)} \left(\frac{\mathcal{L}(\theta') - \mathcal{L}(\theta)}{1 + \mathcal{L}(\theta)}\right)$$

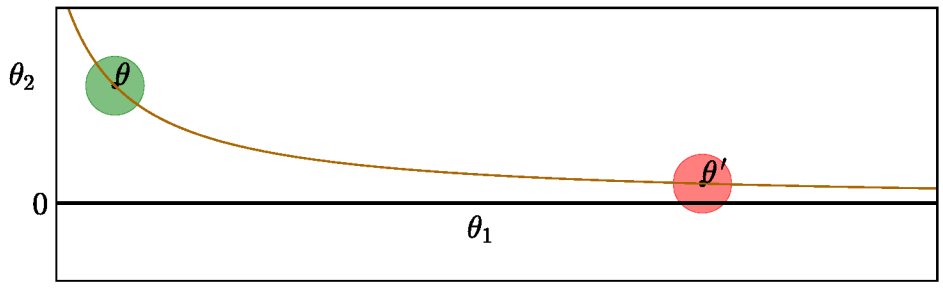


Figure 3.4 – An illustration of how we exploit non-identifiability and its particular geometry to obtain sharper minima: although θ is far from the $\theta_2 = 0$ line, the observationally equivalent parameter θ' is closer. The green and red circle centered on each of these points have the same radius. Best seen with colors.

where $B_2(\epsilon, \theta)$ is the Euclidean ball of radius ϵ centered on θ. This modification will demonstrate more clearly the issues of that metric as a measure of probable generalization. If we use $K = 2$ and (θ_1, θ_2) corresponding to a non-constant function, i.e. $\theta_1 \neq 0$ and $\theta_2 \neq 0$, then we can define $\alpha > \frac{\|\theta_2\|_2}{\epsilon}$. We will now consider the observationally equivalent parameter $T_\alpha(\theta_1, \theta_2) = (\alpha\theta_1, \epsilon\frac{\theta_2}{\|\theta_2\|_2})$. Given that $\|\theta_2\|_2 \leq \|\theta\|_2$, we have that $(\alpha\theta_1, 0) \in B_2\left(\epsilon, T_\alpha(\theta)\right)$, making the maximum loss in this neighborhood at least as high as the loss $\mathcal{L}_{const}$ the best constant-valued function, incurring relatively high sharpness. Figure 3.4 provides a visualization of the proof.

For rectified neural networks, every minimum is observationally equivalent to a minimum that generalizes as well but with high ϵ-sharpness. This also applies when using the actual *full-space* ϵ-sharpness used by Keskar et al. (2017). We can prove this similarly using the equivalence of norms in finite dimensional vector spaces and the fact that for $c > 0, \epsilon > 0, \epsilon \leq \epsilon(c + 1)$ (see Keskar et al. (2017)).

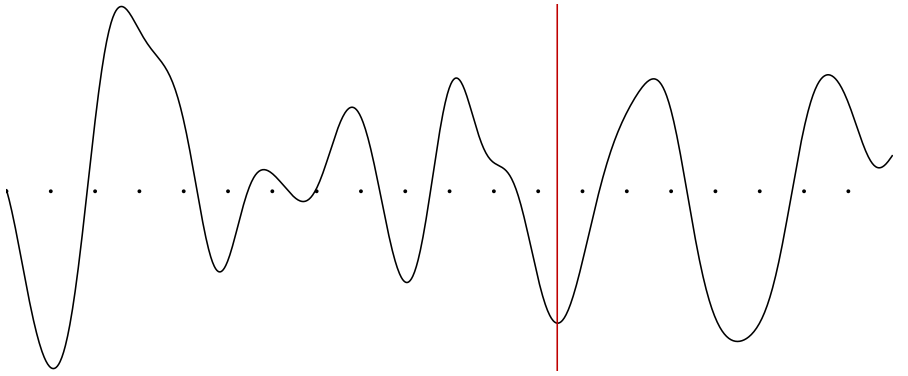
Random subspace This reasoning applies to *random subspace* ϵ-sharpness used by Keskar et al. (2017), i.e. a restriction of the maximization to a random subspace, which relates to the notion of *wide valleys* described by Chaudhari et al. (2017). Indeed, let's take a unitary basis vector u of the random subspace $\mathcal{S}$, u can be uniformly sampled on the sphere. The square of the dot product of u with $\left(0, \frac{\theta_2}{\|\theta_2\|_2}\right)$ will then have a Beta$(\cdot \mid \frac{1}{2}, \frac{d-1}{2})$ distribution. Since it is sufficient to have $\left|(\epsilon\ u) \cdot \left(0, \frac{\theta_2)}{\|\theta_2\|_2}\right)\right| \geq \|\theta_2\|_2$ in order to obtain $\mathcal{L}_{const}$, we have

$$\mathbb{E}_{\mathcal{S}}\left[\max_{\delta\in\mathcal{S}\cap B_2(\epsilon,0)}\left(\frac{\mathcal{L}(\theta+\delta)-\mathcal{L}(\theta)}{1+\mathcal{L}(\theta)}\right)\right] \geq \mathbb{P}\left(\left|(\epsilon\ u)\cdot(0,\theta_2)\right| \geq \|\theta_2\|_2^2\right)\frac{\mathcal{L}_{const}-\mathcal{L}(\theta)}{1+\mathcal{L}(\theta)}$$
$$\Rightarrow \mathbb{E}_{\mathcal{S}}\left[\max_{\delta\in\mathcal{S}\cap B_2(\epsilon,0)}\left(\frac{\mathcal{L}\big(T_\alpha(\theta)+\delta\big)-\mathcal{L}\big(T_\alpha(\theta)\big)}{1+\mathcal{L}\big(T_\alpha(\theta)\big)}\right)\right]$$
$$\geq \mathbb{P}\left(\left|(\epsilon\ u)\cdot(0,\alpha^{-1}\theta_2)\right| \geq \alpha^{-2}\|\theta_2\|_2^2\right)\frac{\mathcal{L}_{const}-\mathcal{L}(\theta)}{1+\mathcal{L}(\theta)} \xrightarrow{\alpha\to+\infty} \frac{\mathcal{L}_{const}-\mathcal{L}(\theta)}{1+\mathcal{L}(\theta)}.$$

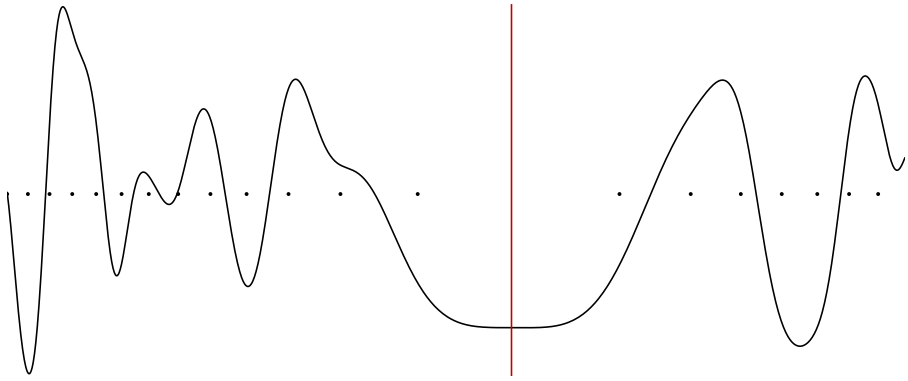
By exploiting the non-Euclidean geometry and non-identifiability of rectified neural networks, we were able to demonstrate some of the limits of using typical definitions of minimum's flatness as core explanation for generalization. Although reaching the degenerate cases built in these proofs might be difficult to reach through standard optimization techniques, it brings up the question of which of the many parameters (and sharpness values) one is using when comparing two predictors in order to assess their relative expected generalization.

3.5 Allowing reparametrizations

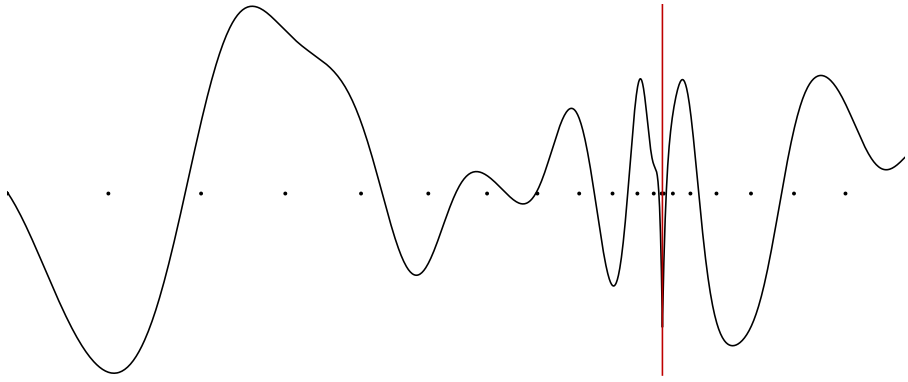
In the previous section 3.4 we explored the case of a fixed parametrization, that of deep rectifier models. In this section we demonstrate a simple observation. If we are allowed to change the parametrization of some function f, we can obtain arbitrarily different geometries without affecting how the function evaluates on unseen data. The same holds for reparametrization of the input space. The implication is that the correlation between the geometry of the parameter space (and hence the error surface) and the behavior of a given function is meaningless if not preconditioned on the specific parametrization of the model.



(a) Loss function with default parametrization



(b) Loss function with reparametrization



(c) Loss function with another reparametrization

Figure 3.5 – A one-dimensional example on how much the geometry of the loss function depends on the parameter space chosen. The x-axis is the parameter value and the y-axis is the loss. The points correspond to a regular grid in the default parametrization. In the default parametrization, all minima have roughly the same curvature but with a careful choice of reparametrization, it is possible to turn a minimum significantly flatter or sharper than the others. Reparametrizations in this figure are of the form $\eta = (|\theta - \hat{\theta}|^2 + b)^a(\theta - \hat{\theta})$ where $b \geq 0, a > -\frac{1}{2}$ and $\hat{\theta}$ is shown with the red vertical line.

3.5.1 Model reparametrization

One thing that needs to be considered when relating flatness of minima to their probable generalization is that the choice of parametrization and its associated geometry are arbitrary. Since we are interested in finding a prediction function in a given family of functions, no reparametrization of this family should influence generalization of any of these functions. Given a bijection Φ onto θ, we can define new transformed parameter $\eta = \Phi^{-1}(\theta)$. Since θ and η represent in different space the same prediction function, they should generalize as well.

Let's call $\mathcal{L}^{(\eta)} = \mathcal{L} \circ \Phi$ the loss function with respect to the new parameter η. We generalize the derivation of subsection 3.4.2:

$$\begin{aligned}
\mathcal{L}^{(\eta)}(\eta) &= \mathcal{L}\big(\Phi(\eta)\big) \\
\Rightarrow (\nabla\mathcal{L}^{(\eta)})(\eta) &= (\nabla\mathcal{L})\big(\Phi(\eta)\big) \cdot (\nabla\Phi)(\eta) \\
\Rightarrow (\nabla^2\mathcal{L}^{(\eta)})(\eta) &= (\nabla\Phi)(\eta)^T \cdot (\nabla^2\mathcal{L})\big(\Phi(\eta)\big) \cdot (\nabla\Phi)(\eta) \\
&\quad + (\nabla\mathcal{L})\big(\Phi(\eta)\big) \cdot (\nabla^2\Phi)(\eta).
\end{aligned}$$

At a differentiable critical point, we have by definition $(\nabla\mathcal{L})\left(\Phi(\eta)\right) = 0$, therefore the transformed Hessian at a critical point becomes

$$(\nabla^2\mathcal{L}^{(\eta)})(\eta) = (\nabla\Phi)(\eta)^T \cdot (\nabla^2\mathcal{L})\big(\Phi(\eta)\big) \cdot (\nabla\Phi)(\eta).$$

This means that by reparametrizing the problem we can modify to a large extent the geometry of the loss function so as to have sharp minima of $\mathcal{L}$ in θ correspond to flat minima of $\mathcal{L}^{(\eta)}$ in $\eta = \Phi^{-1}(\theta)$ and conversely. Figure 3.5 illustrates that point in one dimension. Several practical (see chapter 5, Dinh et al., 2014; Rezende and Mohamed, 2015; Kingma et al., 2016; Dinh et al., 2016) and theoretical works (Hyvärinen and Pajunen, 1999) show how powerful bijections can be. We can also note that the formula for the transformed Hessian at a critical point also applies if Φ is not invertible, Φ would just need to be surjective over Θ in order to cover exactly the same family of prediction functions

$$\{f_\theta, \theta \in \Theta\} = \{f_{\Phi(\eta)}, \eta \in \Phi^{-1}(\Theta)\}.$$

Radial transformation We can describe an elementary transformation to locally perturb the geometry of a finite-dimensional vector space and therefore affect the relative flatness between a finite number minima, at least in terms of spectral norm of the Hessian. We define the function:

$$\begin{aligned}
\forall \delta > 0, \forall \rho \in]0, \delta[, \forall (r, \hat{r}) \in \mathbb{R}_+ \times]0, \delta[, \\
\psi(r, \hat{r}, \delta, \rho) = \mathbb{1}\left(r \notin [0, \delta]\right)\ r + \mathbb{1}\left(r \in [0, \hat{r}]\right)\ \rho\ \frac{r}{\hat{r}} \\
+ \mathbb{1}\left(r \in]\hat{r}, \delta]\right)\ \left((\rho - \delta)\ \frac{r - \delta}{\hat{r} - \delta} + \delta\right) \\
\psi'(r, \hat{r}, \delta, \rho) = \mathbb{1}\left(r \notin [0, \delta]\right) + \mathbb{1}\left(r \in [0, \hat{r}]\right)\ \frac{\rho}{\hat{r}} \\
+ \mathbb{1}\left(r \in]\hat{r}, \delta]\right)\ \frac{\rho - \delta}{\hat{r} - \delta}
\end{aligned}$$

For a parameter $\hat{\theta} \in \Theta$ and $\delta > 0, \rho \in]0, \delta[, \hat{r} \in]0, \delta[$, inspired by the *radial flows* (Rezende and Mohamed, 2015) in we can define the *radial transformations*

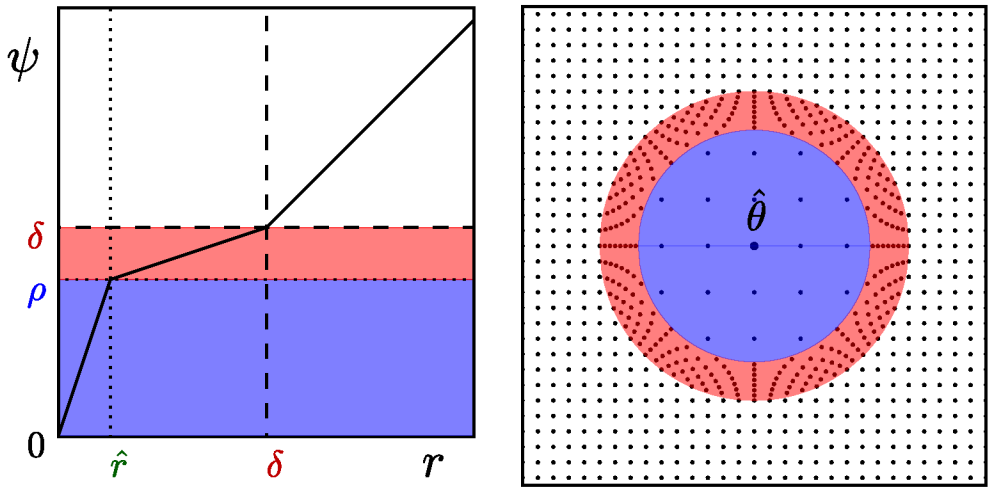
$$\forall \theta \in \Theta,\ \Phi^{-1}(\theta) = \frac{\psi\left(\|\theta - \hat{\theta}\|_2, \hat{r}, \delta, \rho\right)}{\|\theta - \hat{\theta}\|_2}\ \left(\theta - \hat{\theta}\right) + \hat{\theta}$$

with Jacobian

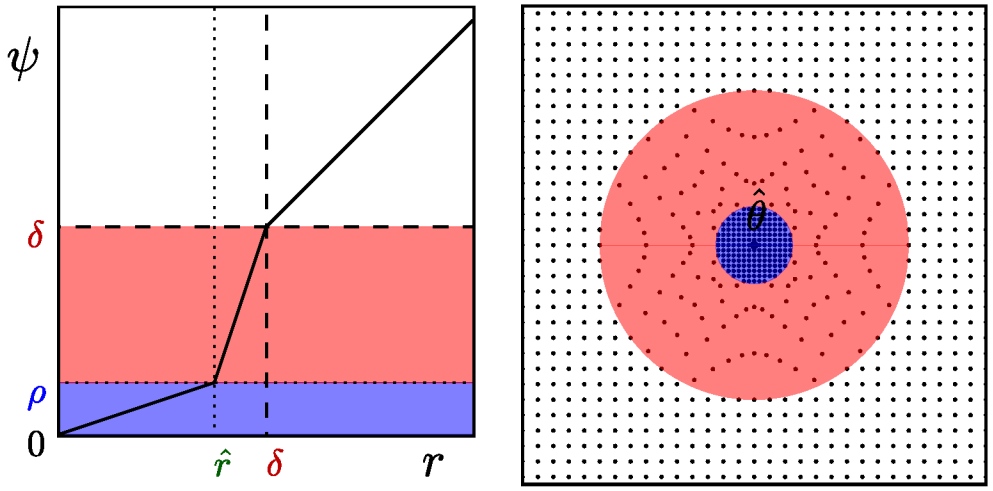
$$\begin{aligned}
\forall \theta \in \Theta,\ (\nabla \Phi^{-1})(\theta) = \ & \psi'(r, \hat{r}, \delta, \rho)\ \mathbb{I}_n \\
& - \mathbb{1}\left(r \in]\hat{r}, \delta]\right) \frac{\delta(\hat{r} - \rho)}{r^3(\hat{r} - \delta)}\ (\theta - \hat{\theta})^T(\theta - \hat{\theta}) \\
& + \mathbb{1}\left(r \in]\hat{r}, \delta]\right) \frac{\delta(\hat{r} - \rho)}{r(\hat{r} - \delta)}\ \mathbb{I}_n,
\end{aligned}$$

with $r = \|\theta - \hat{\theta}\|_2$.

First, we can observe in Figure 3.6 that these transformations are purely local: they only have an effect inside the ball $B_2(\hat{\theta}, \delta)$. Through these transformations, you can arbitrarily perturb the ranking between several minima in terms of flatness. Although this transformation is not differentiable everywhere, neither is a loss function involving rectified networks, and a $\mathcal{C}^\infty$ version can be built from this piece-wise linear version (defined as is for convenience).



(a) $\psi(r, \hat{r}, \delta, \rho)$ **(b)** $\Phi^{-1}(\theta)$



(c) $\psi(r, \hat{r}, \delta, \rho)$ **(d)** $\Phi^{-1}(\theta)$

Figure 3.6 – Two examples of a radial transformation on a 2-dimensional space. We can see that only the area in blue and red, i.e. inside $B_2(\hat{\theta}, \delta)$, are affected. Best seen with colors. In figures 3.6a and 3.6c, the radius transformation, and in figures 3.6b and 3.6d, 2-dimensional visualizations of the transformation.

Normalizations Instances of commonly used reparametrization are *batch normalization* (Ioffe and Szegedy, 2015), or the *virtual batch normalization* variant (Salimans et al., 2016), and *weight normalization* (Badrinarayanan et al., 2015; Salimans and Kingma, 2016; Arpit et al., 2016). Im et al. (2016) have plotted how the loss function landscape was affected by batch normalization. However, we will focus on weight normalization reparametrization as the analysis will be simpler, but the intuition with batch normalization will be similar. Weight normalization reparametrizes a nonzero weight θ_i as $\theta_i = s\frac{\eta_i}{\|\eta_i\|_2}$ with the new parameter being the scale s and the unnormalized weight $\eta_i \neq 0$. Note that without the scale parameter, the *Fisher-Rao norm* (Liang et al., 2017) of η_i,

$$\mathbb{E}_x \left[\langle \eta_i, \nabla_{\eta_i} \ell(\eta_i, x) \rangle^2 \right]$$

is always zero.

Since we can observe that θ_i is invariant to scaling of η_i, reasoning similar to section 3.2 can be applied with the simpler transformations $T'_\alpha : \eta_i \mapsto \alpha\eta_i$ of η_i for $\alpha \neq 0$. Moreover, since this transformation is a simpler isotropic scaling, the conclusion that we can draw can be actually more powerful with respect to η_i:

- every minimum has infinite volume ϵ-sharpness;
- every minimum is observationally equivalent to an infinitely sharp minimum and to an infinitely flat minimum when considering nonzero eigenvalues of the Hessian;
- every minimum is observationally equivalent to a minimum with arbitrarily low full-space and random subspace ϵ-sharpness and a minimum with high full-space ϵ-sharpness.

This further weakens the link between the flatness of a minimum and the generalization property of the associated prediction function, i.e. sharp minima can generalize and flat minima can generalize poorly, when a specific parameter space has not been specified and explained beforehand. In particular, the geometry of default parametrizations used when training deep nets should rarely reflect well the associated generalization properties of the associated minimizers (Li et al., 2017).

3.5.2 Input representation

As we conclude that the notion of flatness for a minimum in the loss function by itself is not sufficient to determine its generalization ability in the general case, we can choose to focus instead on properties of the prediction function instead. Motivated by some work in *adversarial examples* (Szegedy et al., 2014; Goodfellow et al., 2015) for deep neural networks, one could decide on its generalization property by analyzing the gradient of the prediction function on examples. Intuitively, if the gradient is small on typical points from the distribution or has a small Lipschitz constant, then a small change in the input should not incur a large change in the prediction.

But this infinitesimal reasoning is once again very dependent of the local geometry of the input space. For an invertible preprocessing ξ, e.g. *feature normalization* (Aksoy and Haralick, 2001; Coates et al., 2011), *whitening* (Hyvärinen et al., 2004) or *gaussianization* (Chen and Gopinath, 2001), we will call $f_u = f \circ \xi^{-1}$ the prediction function on the preprocessed input $u = \xi(x)$. We can reproduce the derivation in section 3.5 to obtain

$$\frac{\partial f_u}{\partial u^T}\big(\xi(u)\big) = \frac{\partial f}{\partial x^T}\big(\xi^{-1}(u)\big)\frac{\partial \xi^{-1}}{\partial u^T}(u).$$

As we can alter significantly the relative magnitude of the gradient at each point, analyzing the amplitude of the gradient of the prediction function might prove problematic if the choice of the input space have not been explained beforehand. This remark applies in applications involving images, sound or other signals with invariances (Larsen et al., 2015a). For example, Theis et al. (2016) show for images how a small drift of one to four pixels can incur a large difference in terms of L_2 norm.

3.6 Discussion

It has been observed empirically that minima found by standard deep learning algorithms that generalize well tend to be flatter than found minima that did not generalize well (Chaudhari et al., 2017; Keskar et al., 2017). However, when following several definitions of flatness, we have shown that the conclusion that flat

minima should generalize better than sharp ones cannot be applied as is without contextualizing those experimental observations. Previously used definitions fail to account for the complex geometry of some commonly used deep architectures. We demonstrated that by building general enough counter-examples exploiting the *equivariance* of flatness/sharpness with respect to geometry and the *invariance* of generalizability with respect to geometry. In particular, the non-identifiability of the model induced by symmetries, allows one to alter the flatness of a minimum without affecting the function it represents. Additionally the whole geometry of the error surface with respect to the parameters can be changed arbitrarily under different parametrizations.

In the spirit of Swirszcz et al. (2016), this chapter indicates that more care is needed to define flatness to avoid degeneracies of the geometry of the model under study in order to relate it to generalization. Also such a concept can not be divorced from the particular parametrization of the model or input space. For instance, a possible explanation for the observations in Keskar et al. (2017) could be that small batch stochastic gradient descent is more attracted to flat minimizers *in the geometry adopted* and can provide better generalization properties, making the choice of parametrization in stochastic gradient descent a confounding factor. This encourages a more hollistic point of view that includes the learning algorithm in order to obtain a correct and useful interpretation of previous experimental results. This also means that altering the stochastic optimization algorithm could as well change the relationship between loss function landscape and generalization. Alternatively, another possible venue would be to use quantities invariant to reparametrization, as the Kullback-Leibler divergence is, or grounded in a particular reference parametrization and geometry.

4 Deep probabilistic models

Probabilistic modeling is an essential framework in machine learning to take into account uncertainty. Quantifying these uncertainties into *probability distributions* can in turn be used to estimate confidence of in your predictions and assess your potential estimation error. Moreover, considering the stochasticity of the problem can allow the model to disentangle noise from a more learnable correlation structure.

These uncertainties can emerge from different sources. The simplest explanation of these uncertainties is the possible *inherent stochasticity* that we can assume from the problem and its data generating process. Often, this assumption comes from a lack of information, including a *partial observability* of the underlying data generation. For example, we consider a dice throw random because we cannot fully observe the dice throwing process. This lack of information can also come from the inability of a model to fully register every information provided, the approximation made by modeling data are too coarse and the model becomes *underspecified* as a result: some subtlety in the observations can be discarded as random noise.

4.1 Generative models

We will focus on *generative models*, a particular type of probabilistic model, for the rest of the dissertation. These models define an approximation $p_{\theta,X}$ of the data generating process p^*_X and can provide in turn interesting functions and processes. Several of these models enable *density estimation*, possibly conditional, as to estimate confidence in certain outcomes. Synthesis tasks, like speech synthesis, often involve *sampling* from such approximations. That sampling can be conditioned on *clamped* values of a vector in *missing values imputation* problems like *inpainting*.

These models are often learned in an *unsupervised* (or *self-supervised*) fashion. This learning task can become particularly challenging as it involves accounting

for every variability of raw sensory inputs, as opposed to supervised tasks (e.g. classification and regression) where all the information about a data point only needs to be summarized into a single univariate output.

4.1.1 Maximum likelihood estimation

In order to learn these models, one needs to define a cost function to minimize. We choose to train our generative models by minimizing the *Kullback-Leibler divergence* (Kullback and Leibler, 1951) (KL divergence) between the data distribution p_X^* and its approximation p_θ

$$KL(p_X^* \| p_{\theta,X}) = \mathbb{E}_{x \sim p_X^*} \left[\log \left(\frac{p_{\theta,X}(x)}{p_X^*(x)} \right) \right] \geq 0.$$

This divergence belongs to the larger family of reparametrization invariant f-*divergences* (Ali and Silvey, 1966; Csiszár et al., 2004; Liese and Vajda, 2006)

$$D_f(p_X^* \| p_{\theta,X}) = \mathbb{E}_{x \sim p_X^*} \left[f \left(\frac{p_X^*(x)}{p_{\theta,X}(x)} \right) \right] = \mathbb{E}_{\xi(x) \sim p_{\xi(X)}^*} \left[f \left(\frac{p_{\xi(X)}^*(\xi(x))}{p_{\theta,\xi(X)}(\xi(x))} \right) \right],$$

for any function ξ bijective. These f-divergences take 0 as a minimum value and only when $p_X^* = p_{\theta,X}$.

One interesting property is that the problem of minimizing this Kullback-Leibler divergence is equivalent to the consistent *maximum likelihood estimation*, that is

$$\begin{aligned} KL(p_X^* \| p_{\theta,X}) &= \mathbb{E}_{x \sim p_X^*} \left[\log \left(\frac{p_X^*}{p_{\theta,X}} \right) \right] \\ &= \mathbb{E}_{x \sim p_X^*} \left[\log \left(p_X^*(x) \right) \right] - \mathbb{E}_{x \sim p_X^*} \left[\log \left(p_{\theta,X}(x) \right) \right] \\ &= -H(p_X^*) - \mathbb{E}_{x \sim p_X^*} \left[\log \left(p_{\theta,X}(x) \right) \right], \end{aligned}$$

where $H(p_X^*) = -\mathbb{E}_{x \sim p_X^*} \left[\log \left(p_X^*(x) \right) \right]$ is the *(information) entropy* of p^*. As we cannot compute p_X^* in most problems, this entropy cannot be computed. Fortunately, this quantity is constant with respect to $p_{\theta,X}$, therefore minimizing $KL(p_X^* \| p_{\theta,X})$ is equivalent to minimizing

$$\mathcal{L}(p_{\theta,X}, p_X^*) = \mathbb{E}_{x \sim p_X^*} \left[\ell_{NLL} \left(p_{\theta,X}, x \right) \right] \simeq \frac{1}{N} \sum_{x \in \mathcal{D}_{\text{train}}} \ell_{NLL} \left(p_{\theta,X}, x \right) = \hat{\mathcal{L}}(p_{\theta,X}, \mathcal{D}_{\text{train}}),$$

where $\ell_{NLL}\big(p_{\theta,X}, x\big) = -\log\left(p_{\theta,X}(x)\right)$ is the *negative log-likelihood* of $p_{\theta,X}$ on x. This minimization is equivalent to maximizing the *(log-)likelihood*

$$\begin{aligned}\log\left(p_{\theta,X}(\mathcal{D}_{\text{train}})\right) &= \log\left(\prod_{x\in\mathcal{D}_{\text{train}}} p_{\theta,X}(x)\right) \\ &= \sum_{x\in\mathcal{D}_{\text{train}}} \log\left(p_{\theta,X}(x)\right) \\ &= -\sum_{x\in\mathcal{D}_{\text{train}}} \ell_{NLL}\left(p_{\theta,X}, x\right) \qquad .\end{aligned}$$

It is worth noting that several cost functions can find their probabilistic equivalent:

- The squared loss is equivalent to a negative log-likelihood of a Gaussian distribution with fixed variance. Indeed, the log-density of a normal distribution of mean μ and covariance matrix Σ is

  $$\log\left(p_{\mathcal{N}(\cdot|\mu,\Sigma)}(x)\right) = -\frac{1}{2}\left((x-\mu)\Sigma(x-\mu)^T + d_{\mathcal{X}}\log(2\pi) + \log\left(\det(\Sigma)\right)\right).$$

  If $\Sigma = \mathbb{I}_{d_{\mathcal{X}}}$, we recover the square loss up to additive and multiplicative constants with respect to μ.

- The logistic loss with a score s is equivalent to a negative log-likelihood of a Bernoulli distribution $\mathcal{B}(\cdot \mid q)$ with probability $q = \sigma_{sigmoid}(s)$ for $x = 1$. The log-density would be

  $$\log\left(p_{\mathcal{B}(\cdot|q)}(x)\right) = -\left(x\ \log(1+e^{-s}) + (1-x)\log(1+e^{s})\right).$$

4.1.2 Alternative learning principles

Maximum likelihood has been a widely adopted framework for learning probablistic models. There are however several learning principles one could adopt as an alternative to maximum likelihood.

Notably Gutmann and Hyvärinen (2010) introduced the concept of *noise contrastive estimation*, reducing the density estimation problem to a classification problem. In this formulation, the training data is labeled positively ($y = 1$) and is

augmented with an equal set of points generated from a noise distribution $p_{-,X}$, labeled negatively ($y = 0$), resulting in the generating process

$$y \sim \mathcal{B}\left(\cdot \middle| \frac{1}{2}\right)$$
$$\begin{cases} x \mid y & \sim p_X^* \text{if } y = 1 \\ x \mid y & \sim p_{-,X} \text{otherwise.} \end{cases}$$

The task is to classify between data points and noise generated points with a logistic loss. This results in learning an approximation $p_{\theta,Y|X}(y \mid x)$ from which we can estimate the *implicit density* (Mohamed and Lakshminarayanan, 2016)

$$p_{\theta,X}(x) = \frac{p_{\theta,Y|X}(y = 1 \mid x)}{p_{\theta,Y|X}(y = 0 \mid x)} p_{-,X}(x).$$

Designing a proper noise distribution $p_{-,X}(x)$ is essential for applying this learning principle successfully, as the concepts we learn aim at distinguishing both data and noise distributions. The learned structure will be as interesting as the features that distinguish these distributions, too different and the learned structure will be trivial, too similar and the model will only focus on perceptually insignificant details. *Generative adversarial networks* (see Goodfellow et al., 2014b) overcomes this problem by adapting through an adversarial scheme in picking $p_{-,X}(x)$.

Through the lens of noise contrastive estimation, one could see the unsupervised density estimation problem as an extremely unbalanced binary classification task. As unbalanced classification tasks often struggle with the different performance measures, e.g. *precision*, *recall*, or *F-score*, one can use to evaluate those models. This indicates the difficulty of building interesting performance metrics for generative models, in addition to the analyses of Theis et al. (2016). This is why we will consider metrics for this task as probes for properties of our models instead of mere performance metrics, as unsupervised learning remains a misspecified problem [1].

4.1.3 Mixture of Gaussians

To explain the challenges in designing efficient probabilistic generative models, I will introduce the *mixture of Gaussians* model or *Gaussian mixture model*. We

[1] As several machine learning tasks can be.

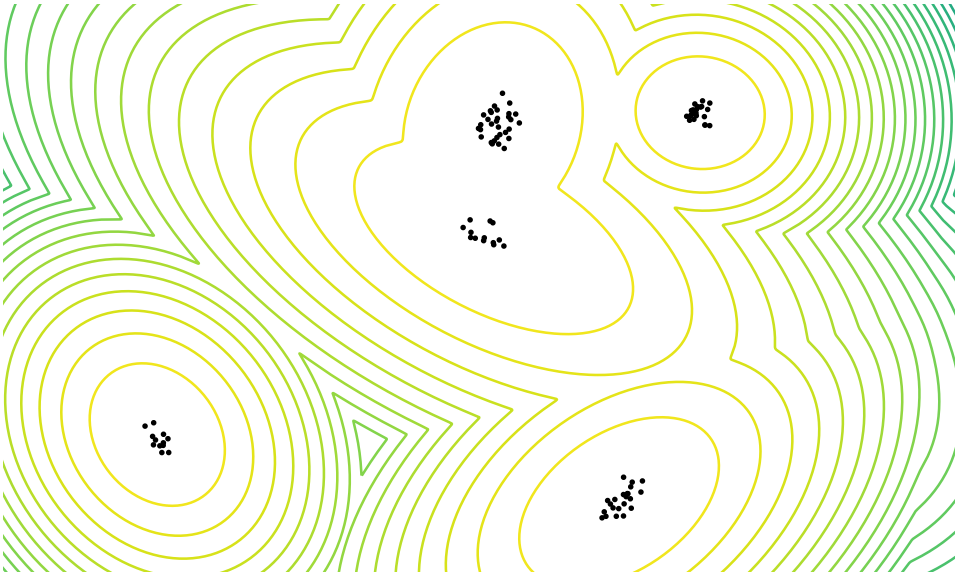


Figure 4.1 – A plot of the log-density of a mixture of Gaussians in a two dimensional space, with samples generated from the same mixture model.

choose $\mathcal{X} = \mathbb{R}^{d_\mathcal{X}}$ and define for K components the following generative process:

$$\begin{aligned} z &\sim \mathrm{Cat}\left(\cdot \mid (\alpha_k)_{k \leq K}\right) \\ x \mid z &\sim \mathcal{N}(\cdot \mid \mu_z, \Sigma_z) \end{aligned}$$

where $(\alpha_k)_{k \leq K} \in [0,1]^K$ such that $\sum_{k=1}^K \alpha_k = 1$ are the mixture coefficients, $\mathrm{Cat}\left(\cdot \mid (\alpha_k)_{k \leq K}\right)$ is the associated categorical distribution, $(\mu_k)_{k \leq K} \in (\mathbb{R}^{d_\mathcal{X}})^K$ are the mixtures means and $(\Sigma_k)_{k \leq K} \in (\mathbb{R}^{d_\mathcal{X} \times d_\mathcal{X}})^K$ are the mixtures covariances.

z is called a *latent* or *unobserved variable* of the model, this latent variable is usually containing more relevant information concerning the associated data point, separated from its noise.

The resulting density model is

$$\begin{aligned} p_{\theta,X}(x) &= \sum_{k=1}^K \alpha_k \, \mathcal{N}(x; \mu_k, \Sigma_k) && \text{(marginal distribution)} \\ p_{\theta,X,Z}(x,z) &= \alpha_z \, \mathcal{N}(x; \mu_z, \Sigma_z) && \text{(joint distribution).} \end{aligned}$$

As there are no known closed form solution for a finite dataset, one can instead optimize iteratively the parameters of the model using the *expectation maximization*

algorithm (Neal and Hinton, 1998; McLachlan et al., 2004). This algorithm involves at every step evaluating the posterior distribution $p_{\theta,Z|X}(z \mid x)$ to evaluate the expected log-distribution

$$\mathbb{E}_{z\sim p_{\theta,Z|X}(\cdot|x)}\left[\log\left(p_{\theta'}(x,z)\right)\right] = \sum_{z=1}^{K} p_{\theta,Z|X}(z \mid x)\log\left(p_{\theta',X,Z}(x,z)\right) \qquad \text{(expectation phase)}$$

and then maximizing the expected log-distribution with respect to θ' (maximization phase) before replacing $\theta \leftarrow \theta'$. Fortunately, this model enables tractable and efficient computation of the posterior distribution $p_{\theta,Z|X}(z \mid x)$ and closed form solution of the problem

$$\arg\max_{\theta'\in\Theta}\left(\mathbb{E}_{z\sim p_{\theta,Z|X}(\cdot|x)}\left[\log\left(p_{\theta',X,Z}(x,z)\right)\right]\right)$$

The Gaussian mixture model assumes an organization of data into clusters and only captures linear correlations within a cluster, meaning that a single mixture component cannot model more complex, nonlinear manifolds. In order to model richer structures, the model has to increase significantly its number of mixture components. However, the number of parameters, and the computational burden for inference and sampling scale linearly with a rapidly increasing number of clusters. In general, the problems of scalable inference, efficient sampling, and parameter efficiency are central when designing generative models.

4.2 Deep generative models

In order to efficiently capture the complexity of interesting problems, researchers capitalized on the progress of machine learning models in supervised learning and the increase of computational power. In particular, they harnessed the recent advances in deep learning (presented in chapter 2) to build deep generative models.

An important challenge when implementing deep generative models is enabling tractability of their various operations, sampling, inference, and density estimation, despite the highly nonlinear nature of deep models of interest.

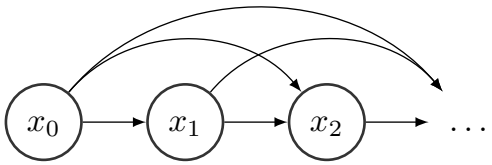


Figure 4.2 – A probabilistic graphical model corresponding to autoregressive model. This graph represent a trivial connectivity, it is fully connected.

4.2.1 Autoregressive models

A simple way to design generative models with deep neural networks stems from the observation that deep models can already model conditional probabilities. As we have pointed out in subsection 4.1.1, one can for example build a conditional scalar Gaussian distribution by outputting a mean and a variance parameters for this conditional distribution. In general, a conditional distribution can be implemented using a deep network by outputting the parameters of this distribution.

Then, a joint distribution can be implemented with neural networks using a simple probability chain rule (Frey, 1998)

$$p_{\theta,X}(x) = p_{\theta,X_1}(x_1) \prod_{k=1}^{d_{\mathcal{X}}-1} p_{\theta,X_{k+1}|X_{<k}}(x_{k+1} \mid x_{<k}),$$

with $p_{\theta,X_{k+1}|X_{<k}}$ defined as a conditional distribution on x_{k+1} whose parameters is an output of a neural network with input $(x_{k'})_{<k}$. This decomposition correspond to a trivial dependency structure and *probabilistic graphical model* (see figure 4.2).

A very inefficient way to implement these neural networks would be to create $d_{\mathcal{X}}-1$ separate neural networks for each conditional model $p_{\theta,X_{k+1}|X_{<k}}(x_{k+1} \mid x_{k'<k})$. We can easily show that that solution hardly scales with the dimensionality of the problem. Bengio and Bengio (1999); Larochelle and Murray (2011) propose instead to reuse computations and reduce parameters by using a single neural network with a specific sparsity pattern in connectivity (see figure 4.3) such that only the input $(x_{k'})_{k'<k}$ can define the distribution for x_k. Such topology can be efficiently implemented through the use of recurrent neural networks (e.g. Bengio et al., 2003), for example long short term memory networks (e.g. Graves, 2013; Theis and Bethge, 2015; van den Oord et al., 2016b), or by adopting an *autoregressive masking patterns* (Germain et al., 2015; van den Oord et al., 2016b,a).

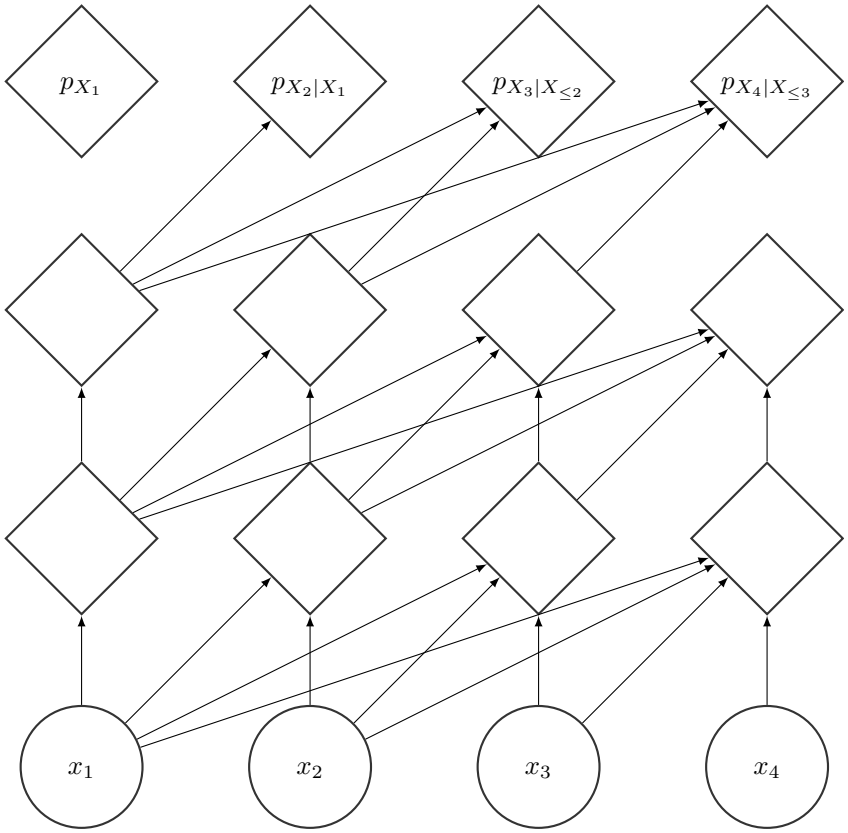


Figure 4.3 – An example of autoregressive masking pattern, the computational graph represented going bottom up. We see that there is no path connecting x_k to $p(x_k \mid x_{<k})$.

Using that learning scheme has many advantages. The learning of the model can be clearly done through stochastic gradient descent on a tractable negative log-likelihood criterion, and without the presence of latent variables, inference can be done very efficiently. The graph structure of the model reveals a very straightforward unbiased *ancestral sampling* procedure to generate from it.

$$
\begin{aligned}
x_1 &\sim p_{X_1} \\
x_2 &\sim p_{X_2|X_1}(\cdot \mid x_1) \\
&\dots \\
x_{d_\mathcal{X}} &\sim p_{X_{d_\mathcal{X}}|X_{<d_\mathcal{X}}}(\cdot \mid x_{<d_\mathcal{X}})
\end{aligned}
$$

Nonetheless, one could hope to improve on the sampling procedure, which scales linearly with the number of dimension of the input space and is non-parallelizable.

4.2.2 Deep generator networks

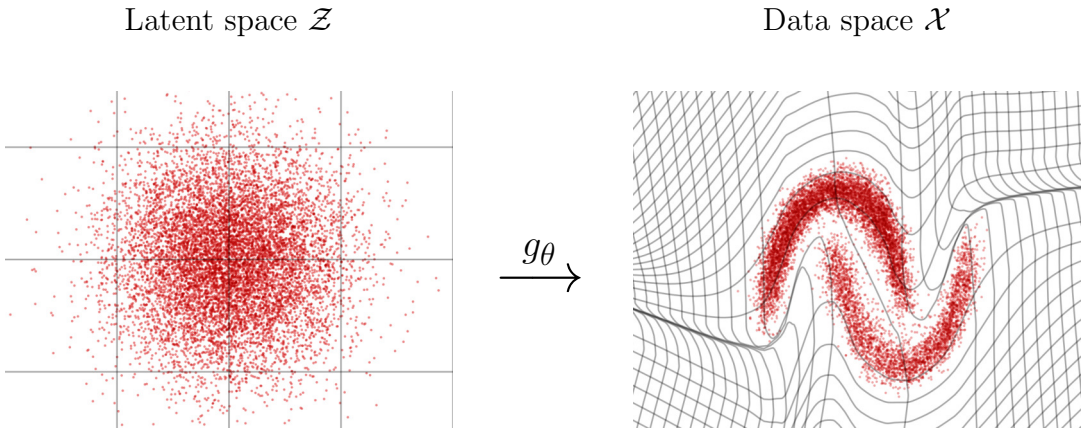


Figure 4.4 – A generator network approach follows the spirit of inverse transform sampling: sample from a simple distribution a noise variable z, and pass it through the generator network g_θ to obtain a sample $x = g_\theta(z)$ with the desired distribution. In this example, the desired distribution is the *two moons dataset*, which exhibit two interesting properties in its structure: the separation into two clusters, and the nonlinear correlation structure among each cluster.

Given a cumulative distribution function CDF_p, *inverse transform sampling* (Devroye, 1986) is a scheme to efficiently sample from a univariate distribution p

$$u \sim \mathcal{U}([0,1]) \tag{4.1}$$

$$x = \mathrm{CDF}_p^{-1}(u). \tag{4.2}$$

This decomposes the problem of sampling from a complex distribution into sampling from a simple uniform distribution and the application of an inverse CDF.

In a similar fashion for higher dimensionalities, the deep generator network approach to generative models aims at building a generative process that can be summarized into two elementary steps of

- sampling $z \in \mathcal{Z} = \mathbb{R}^{d_Z}$ from a simple standard distribution p_Z (see equation 4.1);
- passing that sample through a deep neural network g_θ (see equation 4.2 and figure 4.4).

These operations can often be well parallelized.

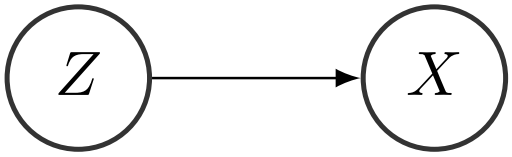


Figure 4.5 – Graphical model corresponding to the emission model: a latent noise variable z is generated from a standard distribution, this variable is then used to build the parameters of the conditional distribution $p_{\theta,X|Z}$.

Emission model

One potential issue when attempting to learn deep generator network models with maximum likelihood is that their associated density model might not be well defined everywhere or just 0, especially if $\mathcal{Z}$ has a lower dimensionality than $\mathcal{X}$. A typical solution to this is to add noise to the resulting distribution.

The resulting generative process remains very simple as illustrated figure 4.5: we sample z typically from a standard distribution p_Z, e.g. $\mathcal{N}(\cdot \mid 0, \mathbb{I}_{d_\mathcal{Z}})$, then pass z through a neural network to output the parameters of a conditional distribution $p_{\theta,X|Z}$, e.g. $\mathcal{N}(\cdot \mid \mu_\theta(z), \Sigma_\theta(z))$.

Variational inference

Following the standard approach to train deep models, deep probabilistic models can be learned using gradient-based methods. When learning deep latent variable models, the differentiation of the log-likelihood goes as follow

$$\begin{aligned}
\frac{\partial \log\left(p_{\theta,X}(x)\right)}{\partial \theta^T} &= \frac{1}{p_{\theta,X}(x)} \frac{\partial p_{\theta,X}(x)}{\partial \theta^T} \\
&= \frac{1}{p_{\theta,X}(x)} \frac{\partial}{\partial \theta^T} \left(\mathbb{E}_{z \sim p_Z}\left[p_{\theta,X|Z}(x \mid z)\right]\right) \\
&= \frac{1}{p_{\theta,X}(x)} \frac{\partial}{\partial \theta^T} \left(\int_{z \in \mathcal{Z}} p_{\theta,X|Z}(x \mid z) p_Z(z) dz\right) \\
&= \frac{1}{p_{\theta,X}(x)} \int_{z \in \mathcal{Z}} \frac{\partial}{\partial \theta^T} \left(p_{\theta,X|Z}(x \mid z)\right) p_Z(z) dz \\
&= \frac{1}{p_{\theta,X}(x)} \int_{z \in \mathcal{Z}} p_{\theta,X|Z}(x \mid z) \frac{\partial \log\left(p_{\theta,X|Z}(x \mid z)\right)}{\partial \theta^T} p_Z(z) dz
\end{aligned}$$

$$
\begin{aligned}
&= \int_{z\in\mathcal{Z}} \frac{p_{\theta,X|Z}(x \mid z)p_Z(z)}{p_{\theta,X}(x)} \cdot \frac{\partial \log\left(p_{\theta,X|Z}(x \mid z)\right)}{\partial \theta^T} dz \\
&= \int_{z\in\mathcal{Z}} \frac{p_{\theta,X,Z}(x,z)}{p_{\theta,X}(x)} \cdot \frac{\partial \log\left(p_{\theta,X|Z}(x \mid z)\right)}{\partial \theta^T} dz \\
&= \int_{z\in\mathcal{Z}} \frac{\partial \log\left(p_{\theta,X|Z}(x \mid z)\right)}{\partial \theta^T} \, p_{\theta,Z|X}(z \mid x) dz \\
&= \mathbb{E}_{z\sim p_{\theta,Z|X}} \left[\frac{\partial \log\left(p_{\theta,X|Z}(x \mid z)\right)}{\partial \theta^T}\right].
\end{aligned}
$$

This means that even for gradient descent on log-likelihood, computing the posterior $p_{\theta,Z|X}$ is necessary. However, unlike with Gaussian mixture models, the problem of computing the posterior $p_{\theta,Z|X}$ becomes intractable for deep latent variable models as it involves computing for instance the expectation $p_{\theta,X}(x) = \mathbb{E}_{z\sim p_Z}\left[p_{\theta,X|Z}(x \mid z)\right]$. Another reason for this intractability is the well known *explaining away* effect: conditioned on an observed variable x, the elements z_i of the random vector z, that were independent unconditionally, become deeply entangled.

A typical approach to overcome this intractability is to use instead an approximation q of the posterior $p_{\theta,Z|X}$. This *approximate posterior* will usually be designed to allow simpler computation of expectations involved, or at least a Monte Carlo approximation of them. Learning using this approximate posterior is enabled by the use of the *variational lower bound* (Neal and Hinton, 1998; Wainwright et al., 2008), also known as the *evidence lower bound*,

$$
\begin{aligned}
\log(p_{\theta,X}(x)) &= \int_z \log(p_{\theta,X}(x))q(z)dz \\
&= \int_z \log\left(\frac{p_{\theta,X,Z}(x,z)}{p_{\theta,Z|X}(z \mid x)}\right) q(z)dz \\
&= \int_z \log\left(\frac{p_{\theta,X,Z}(x,z)q(z)}{p_{\theta,Z|X}(z \mid x)q(z)}\right) q(z)dz \\
&= \int_z \log\left(\frac{p_{\theta,X,Z}(x,z)}{q(z)}\right) q(z)dz + \int_z \log\left(\frac{q(z \mid x)}{p_{\theta,Z|X}(z \mid x)}\right) q(z)dz \\
&= \mathbb{E}_{z\sim q(z)}\left[\log\left(\frac{p_{\theta,X,Z}(x,z)}{q(z)}\right)\right] + KL\left(q(z) \| p_{\theta,Z|X}(z \mid x)\right) \\
&\geq \mathbb{E}_{z\sim q(z)}\left[\log\left(\frac{p_{\theta,X,Z}(x,z)}{q(z)}\right)\right] = -\mathcal{L}_{ELBO}(p_{\theta,X,Z}, q, x).
\end{aligned}
$$

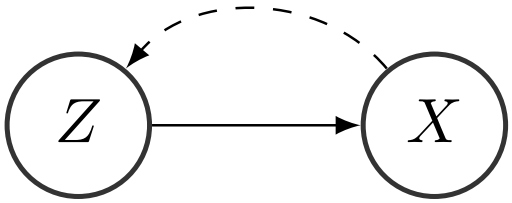


Figure 4.6 – The Helmoltz machine uses an inference network to output an approximate posterior for variational inference $q_\lambda(\cdot \mid x)$.

The choice of q will typically tradeoff the approximation quality, quantified by $KL\left(q(z) \| p_{\theta,Z|X}(z \mid x)\right)$, for computational tractability. A typical approximation one uses is the *mean field approximation* (Saul et al., 1996), where all the dimensions of z given x are considered independent. For a Gaussian distribution, it means that the covariance matrix will be diagonal.

Gradient-based inference

Training a deep latent variable model with an approximate posterior depends on the approximation quality of the variational lower bound. This depends on how close q is to $p_{\theta,Z|X}$. While *undirected models*, such as *restricted Boltzmann machines* (Smolensky, 1986) and *deep Boltzmann machines* (Salakhutdinov and Hinton, 2009) benefits from a convenient conditional independence structure that allows *fixed point iteration* inference, choosing a parametrized distribution q_λ will be one of the few pratical options in this case (e.g. Valpola and Karhunen, 2002; Honkela and Valpola, 2004). This parametrized approximate posterior can be chosen so as to minimize the divergence $KL\left(q_\lambda(z) \| p_{\theta,Z|X}(z \mid x)\right)$ or, equivalently, maximize the variational lower bound. This reframes the problem of inference into an optimization problem for every x of the variational lower bound, which becomes the unified cost function for optimizing both generation and inference. *Stochastic variational inference* (Paisley et al., 2012; Hoffman et al., 2013) involves estimating this bound through a Monte Carlo scheme

$$\log\left(\frac{p_{\theta,X,Z}(x,z)}{q_\lambda(z)}\right) \qquad\qquad z \sim q_\lambda.$$

In order to *amortize* the cost of *inference* for each x and θ encountered, *Helmholtz machines* (Dayan et al., 1995) exploit the commonalities between inference prob-

lems over each x by learning a parametrized *inference network* $q_\lambda(\cdot \mid x)$ (see figure 4.6) where the parameter of the distribution is a function of x. For instance, if the approximate posterior is a diagonal Gaussian then the inference network will output its mean and standard deviation $\mu_\lambda(x), \sigma_\lambda(x)$. The resulting model becomes very similar to a *regularized autoencoder* (Hinton and Zemel, 1993) where z can be seen as a stochastic code and the loss function

$$-\mathbb{E}_{z\sim q(z)}\left[\log\left(\frac{p_{\theta,X,Z}(x,z)}{q_\lambda(z \mid x)}\right)\right] = -\mathbb{E}_{z\sim q(z)}\left[\log\left(p_{\theta,X|Z}(x \mid z)\right)\right] + KL\left(q_\lambda(z \mid x)\|p_Z(z)\right)$$

can be decomposed into a reconstruction error and a regularization term on the stochastic code. This comparison have earned this model the name *variational autoencoder* (Kingma and Welling, 2014a; Mnih and Gregor, 2014; Rezende et al., 2014), with q_λ as the encoder and $p_{\theta,X|Z}$ as the decoder.

This inference network will be typically learned through gradient descent as well, which will be *doubly stochastic* as both z and x are sampled, x from the dataset and z from $q_\lambda(\cdot \mid x)$. A widely applicable approach to obtain an unbiased gradient estimator is the use of the REINFORCE (REward Increment = Non-negative Factor $\times$ Offset Reinforcement $\times$ Characteristic Eligibility) estimator (Williams, 1992). This estimator, also known as the *score function estimator* (Kleijnen and Rubinstein, 1996) or *likelihood ratio estimator* (Glynn, 1990), for the variational lower bound is obtained by the following derivation:

$$\begin{aligned}
&-\frac{\partial \mathcal{L}_{ELBO}\left(p_{\theta,X,Z}, q_\lambda, x\right)}{\partial \lambda^T} \\
&\quad= \frac{\partial}{\partial \lambda^T}\left(\mathbb{E}_{z\sim q(z)}\left[\log\left(\frac{p_{\theta,X,Z}(x,z)}{q_\lambda(z \mid x)}\right)\right]\right) \\
&\quad= \frac{\partial}{\partial \lambda^T}\left(\int_{z\in\mathcal{Z}} \log\left(\frac{p_{\theta,X,Z}(x,z)}{q_\lambda(z \mid x)}\right) q_\lambda(z \mid x) dz\right) \\
&\quad= \int_{z\in\mathcal{Z}} \frac{\partial}{\partial \lambda^T}\left(\log\left(\frac{p_{\theta,X,Z}(x,z)}{q_\lambda(z \mid x)}\right) q_\lambda(z \mid x)\right) dz \\
&\quad= \int_{z\in\mathcal{Z}} \left(\log\left(\frac{p_{\theta,X,Z}(x,z)}{q_\lambda(z \mid x)}\right) \frac{\partial q_\lambda(z \mid x)}{\partial \lambda^T} - \frac{\partial \log\left(q_\lambda(z \mid x)\right)}{\partial \lambda^T} q_\lambda(z \mid x)\right) dz
\end{aligned}$$

$$= \int_{z \in \mathcal{Z}} \log \left(\frac{p_{\theta,X,Z}(x,z)}{q_\lambda(z \mid x)} \right) \frac{\partial \log \big(q_\lambda(z \mid x)\big)}{\partial \lambda^T} q_\lambda(z \mid x) dz$$
$$= \mathbb{E}_{z \sim q(z)} \left[\log \left(\frac{p_{\theta,X,Z}(x,z)}{q_\lambda(z \mid x)} \right) \frac{\partial \log \big(q_\lambda(z \mid x)\big)}{\partial \lambda^T} \right]$$

estimated by

$$\log \left(\frac{p_{\theta,X,Z}(x,z)}{q_\lambda(z \mid x)} \right) \frac{\partial \log \big(q_\lambda(z \mid x)\big)}{\partial \lambda^T} \qquad\qquad z \sim q_\lambda(\cdot \mid x).$$

This estimator suffers however from high variance, hindering its efficiency for training any models. Several *variance reduction* techniques attemping to address this issue were developed, including *control variate* approaches (e.g. Greensmith et al., 2004; Paisley et al., 2012; Tucker et al., 2017; Grathwohl et al., 2017). One particularly efficient solution for variational autoencoders was the use of the *reparametrization trick* (Williams, 1992; Graves, 2011; Bengio et al., 2014; Kingma and Welling, 2014a,b; Rezende et al., 2014). For a continuous latent variable z, one can use the principle of inverse transform sampling to re-express z as a function of an *auxiliary random variable* ϵ (see figure 4.7) with standard distribution p^*_ϵ. For example, if $z \sim \mathcal{N}\big(\cdot \mid \mu_\lambda(x), \sigma_\lambda(x)\big)$, then we can rewrite

$$z = \mu_\lambda(x) + \sigma_\lambda(x) \odot \epsilon$$

with $\epsilon \sim \mathcal{N}(\cdot \mid 0, \mathbb{I}_{d_\mathcal{Z}})$ and $\odot$ being the element-wise multiplication. The gradient can then be simply estimated using backpropagation and the *law of the unconscious statistician* by

$$\frac{\partial \log \big(p_{\theta,X|Z}(x \mid \mu_\lambda(x) + \sigma_\lambda(x))\big)}{\partial \lambda^T} + \frac{\partial KL\big(q_\lambda(z \mid x) \| p_Z(z)\big)}{\partial \lambda^T} \qquad \epsilon \sim \mathcal{N}(\cdot \mid 0, \mathbb{I}_{d_\mathcal{Z}}).$$

Although this estimator is not guaranteed to be lower variance than the score function estimator (Gal, 2016), it has often proven to be the most practical for training deep generative models with continuous latent variables.

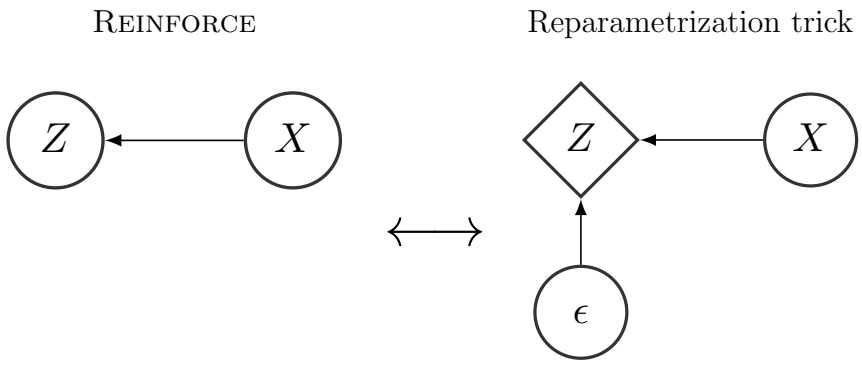


Figure 4.7 – An illustration of the reparametrization trick. Considering z as a random variable sampled from a conditional distribution $q_\lambda(z \mid x)$ (e.g. $\mathcal{N}(\cdot \mid \mu_\lambda(x), \sigma_\lambda(x))$) results in using the REINFORCE estimator. However, by rewriting z as deterministic function of x and a stochastic standard variable ϵ (e.g. $\mu_\lambda(x) + \sigma_\lambda(x)$ with $z \sim \mathcal{N}(\cdot \mid 0, \mathbb{I}_{d_Z})$), we can estimate a potentially lower variance gradient estimate by backpropagation.

Evaluation

Although these variational autoencoders can be successfully trained to maximize the evidence lower bound, the lower bound obtained with a mean field approximation remains an inaccurate estimate of the log-likelihood. This results in some degree of underfitting, which results in practice in models that generate blurry samples (for fully connected models) or samples that lack global coherency (for convolutional models). Moreover, the quality of this approximation, quantified by $KL\left(q_\lambda(z \mid x) \| p_Z(z)\right)$, is also intractable.

A more accurate lower bound can be provided using *importance sampling* (Rezende et al., 2014; Burda et al., 2016):

$$
\begin{aligned}
&\log\big(p_{\theta,X}(x)\big) \\
&\quad = \log\left(\mathbb{E}_{z\sim p_Z}\left[p_{\theta,X|Z}(x \mid z)\right]\right) \\
&\quad = \log\left(\mathbb{E}_{z\sim q_\lambda(\cdot|x)}\left[p_{\theta,X|Z}(x \mid z)\frac{p_Z(z)}{q_\lambda(z \mid x)}\right]\right) \\
&\quad = \log\left(\mathbb{E}_{z\sim q_\lambda(\cdot|x)}\left[\frac{p_{\theta,X,Z}(x,z)}{q_\lambda(z \mid x)}\right]\right) \\
&\quad = \log\left(\mathbb{E}_{(z_k)_{k\leq K}\sim q_\lambda(\cdot|x)^K}\left[\frac{1}{K}\sum_{k=1}^{K}\frac{p_{\theta,X,Z}(x,z_k)}{q_\lambda(z_k \mid x)}\right]\right)
\end{aligned}
$$

$$\geq \mathbb{E}_{(z_k)_{k\leq K}\sim q_\lambda(\cdot|x)^K}\left[\log\left(\frac{1}{K}\sum_{k=1}^{K}\frac{p_{\theta,X,Z}(x,z_k)}{q_\lambda(z_k\mid x)}\right)\right] \xrightarrow{K\to+\infty} \log\big(p_{\theta,X}(x)\big).$$

This bound has the advantage to be asymptotically unbiased with respect to the log-likelihood, however Wu et al. (2017) empirically showed that the rate of convergence remains too slow to be accurate enough in practice.

5 Real NVP: a deep tractable generative model

Ideally, we would want to capitalize on the advantage of fast generative process that the generator network paradigm provides while being able to evaluate the model performance accurately and efficiently. Such a generative model should also be expressive enough so as to enable accurate enough modeling data with rich structure. This final chapter will present the nonlinear independent components estimation (NICE) (Dinh et al., 2014) approach, for *real-valued non-volume preserving* with REAL NVP (Dinh et al., 2016), which provides a possible avenue to explore that approach.

5.1 Computing log-likelihood

5.1.1 Change of variable formula

In order to build that model, we can look back at the inverse transform sampling algorithm described subsection 4.2.2 (see figure 5.1). The principle that underpins this approach is the *change of variable formula*, which provides a closed form expression of the density of $g_\theta(z)$

$$p_{\theta,X}\left(g_\theta(z)\right) = p_Z(z)\left|\det\left(\frac{\partial g_\theta(z)}{\partial z^T}\right)\right|^{-1},$$

with $d_{\mathcal{Z}} = d_{\mathcal{X}}$ and g_θ bijective from $\mathcal{Z}$ to $\mathcal{X}$. This formula (which can be derived from a $p(x)dx = p(z)dz$ identity), which quantifies the change in density incurred through bijective reparametrizations, can be understood as follow:

Work (Dinh et al., 2014, 2016) done in collaboration with Dr. Jascha Sohl-Dickstein, David Krueger, Dr. Samy Bengio, and Pr. Yoshua Bengio. Part of this work (Dinh et al., 2016) was done during an internship at Google Brain.

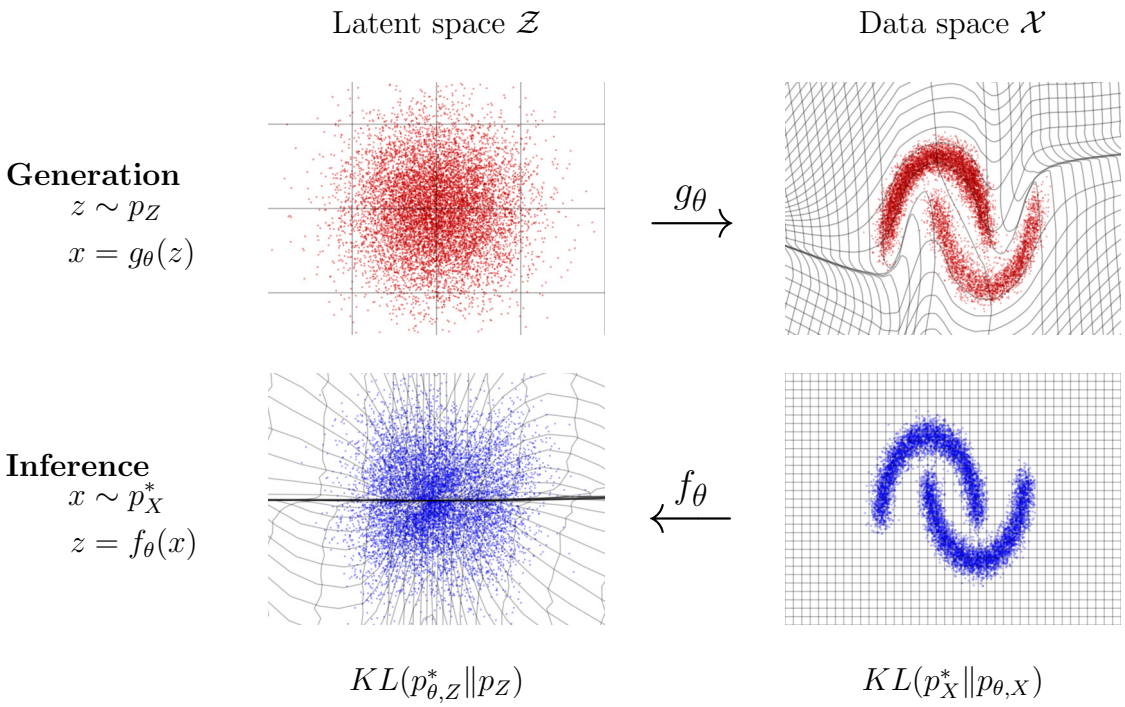


Figure 5.1 – Real NVP follows the generator network paradigm (first line with red figures): g_θ transforms a prior distribution (here standard Gaussian) into an approximation of the data distribution. The training of the model is very similar to expectation maximization as the function f_θ provides the inference for z given a data point x (second line with blue figures). However, the generator function here is not only deterministic but also invertible (bijective to be more precise): there is only one latent vector z corresponding to x. The transformations are demonstrated on a model trained on the two moons dataset. Best seen with colors.

- the density of the outcome $g_\theta(z)$ is proportional to $p_Z(z)$ the prior density in the latent space;
- the Jacobian $\frac{\partial g_\theta(z)}{\partial z^T}$ quantifies the local infinitesimal changes obtained by applying the generator network function g_θ. In particular, the Jacobian determinant $\left|\det\left(\frac{\partial g_\theta(z)}{\partial z^T}\right)\right|$ reflects the local change of volume which affects the density as

$$\text{density} = \frac{\text{mass}}{\text{volume}}.$$

The maximum likelihood formulation of *independent components analysis* (Pham and Garat, 1997; Hyvärinen et al., 2004; Hyvärinen and Pajunen, 1999) is built on the change of variable formula

$$p_{\theta,X}(x) = p_Z(f_\theta(x)) \left|\det\left(\frac{\partial f_\theta(x)}{\partial x^T}\right)\right|,$$

where $f_\theta = g_\theta^{-1}$ is the *unmixing function* or *normalizing flow* (Tabak and Turner, 2013) and with p_Z a distribution with independent components, *gaussianization* (Chen and Gopinath, 2001) being a special case where p_Z is a standard Gaussian distribution. For log-likelihood, the formula becomes

$$\log\big(p_{\theta,X}(x)\big) = \log\big(p_Z\big(f_\theta(x)\big)\big) + \log\left(\left|\det\left(\frac{\partial f_\theta(x)}{\partial x^T}\right)\right|\right),$$

Evaluating this density $p_Z\big(f_\theta(x)\big)$ is generally easy as it involves a forward pass through a function f_θ, that we will could parametrize as a feedforward network, and p_Z can be chosen to be as simple as a standard Gaussian. However, computing the Jacobian determinant $\det\left(\frac{\partial f_\theta(x)}{\partial x^T}\right)$ can quickly become intractable in high dimensional cases, as computing the Jacobian of a function is expensive and so is computing the determinant of the resulting large Jacobian matrix. Moreover, an additional hurdle when using that model for generation is computing a forward pass through generator network $g_\theta = f_\theta^{-1}$, which requires inverting a non-trivial f_θ function.

While direct computation of these quantities is still manageable in low dimensional problems (e.g Bengio, 1991; Baird et al., 2005), several works (e.g. Valpola and Karhunen, 2002; Honkela and Valpola, 2004; Rippel and Adams, 2013) resorted to use an emission model instead as described in 4.2.2, trained through variational inference.

5.1.2 Tractable architecture

Design of invertible functions

One of the challenges when training these invertible generative models is to design architectures that enable these computations. Autoregressive models (described in subsection 4.2.1) are already allowing tractable log-likelihood computation and sampling, albeit by a trivial chain rule probability decomposition.

Autoregressive invertible functions Hyvärinen and Pajunen (1999) offers an interesting reformulation of these autoregressive models as a nonlinear bijective transformation: if we consider the conditional cumulative distribution functions

$\mathrm{CDF}_{\theta,X_k|X_{<k}}$ associated to the density $p_{\theta,X}$, then

$$z_k = \mathrm{CDF}_{\theta,X_k|X_{<k}}(x_k \mid x_{<k})$$

are independent with distribution $\mathcal{U}([0,1])$ if $x \sim p_{\theta,X}$. Equivalently, if $z \sim \mathcal{U}([0,1])^{d_{\mathcal{Z}}}$ then

$$\begin{aligned} & & x_1 &= \mathrm{CDF}^{-1}_{\theta,X_1}(z_1) \\ \forall k \leq d_{\mathcal{Z}}, & & x_k &= \mathrm{CDF}^{-1}_{\theta,X_k|X_{<k}}(z_k \mid x_{<k}). \end{aligned}$$

This view reframes successfully autoregressive models into the nonlinear independent components analysis framework. Nonetheless, the inversion process can only be done dimension by dimension and is hardly parallelizable.

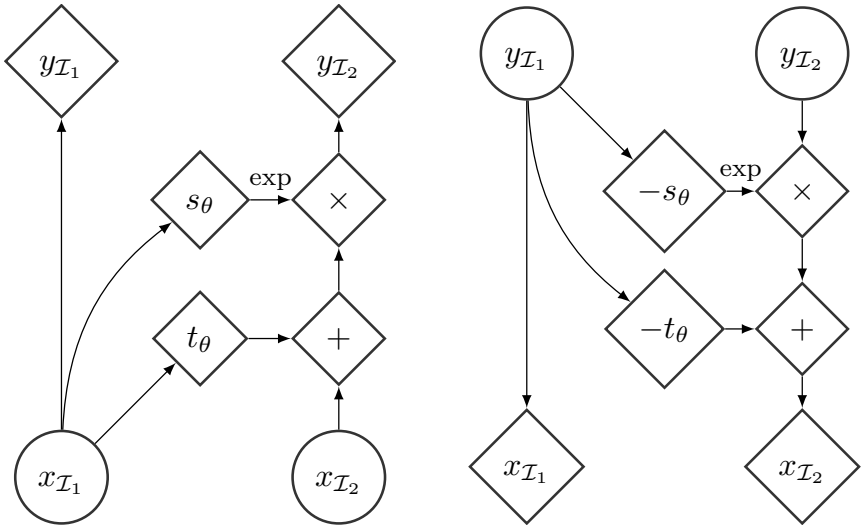
Coupling layer We can notice that the linear computational cost with the number of dimensions is actually due to the fine-grained partitioning of the dimensions into individual components $(x_1, x_2, \ldots, x_{d_{\mathcal{X}}})$. However, one could exploit a coarser partioning $(\mathcal{I}_1, \mathcal{I}_2)$ that would allow for better parallelization. To that effect, we proposed to use the following *affine coupling layer* structure (see figure 5.2a)

$$\begin{aligned} y_{\mathcal{I}_1} &= x_{\mathcal{I}_1} \\ y_{\mathcal{I}_2} &= \big(x_{\mathcal{I}_2} + t_\theta(x_{\mathcal{I}_1})\big) \odot \exp\big(s_\theta(x_{\mathcal{I}_1})\big). \end{aligned}$$

A forward pass through such layer leaves the dimensions in $\mathcal{I}_1$ unchanged whereas $x_{\mathcal{I}_2}$ is transformed linearly with parameters output by *translation* and *scaling functions* $(t_\theta, s_\theta) \in \big(\mathbb{R}^{\mathcal{I}_1} \mapsto \mathbb{R}^{\mathcal{I}_2}\big)^2$. t_θ and s_θ have little constraints on their definition apart from their interfaces, the domain and the codomain. In our experiments, we will choose them to be the output of a deep neural network. In a fashion similar to Germain et al. (2015); van den Oord et al. (2016b,a), this partitioning can be rewritten using a binary masking b, with $b_{\mathcal{I}_1} = 1$ and $b_{\mathcal{I}_2} = 0$,

$$\begin{aligned} y &= b \odot x + (1-b) \odot \big(x + t_\theta(b \odot x)\big) \odot \exp\big(s_\theta(b \odot x)\big) \\ &= \big(x + (1-b) \odot t_\theta(b \odot x)\big) \odot \exp\big((1-b) \odot s_\theta(b \odot x)\big). \end{aligned}$$

When $s_\theta = 0$ we will call the coupling layer *additive*.



(a) Computational graph for the forward pass of an affine coupling layer.

(b) Computational graph for the forward pass of the inverse of an affine coupling layer.

Figure 5.2 – Computational graphs (from bottom up) corresponding to the forward pass of an affine coupling layer and its inverse.

This model can be seen as a neural version of *Feistel networks* (Menezes et al., 1996), used in cryptography, or a nonlinear *lifting scheme* (Sweldens, 1998), used in signal processing.

Properties of coupling layers

Although the proposed architecture is unconventional in deep learning, its properties make it very practical for the purpose of learning a deep generator network.

Inversion Learning and sampling from a generator network through maximum likelihood requires being able to invert its forward pass. A coupling layer follows a pattern similar to a *lifting scheme* (Sweldens, 1998) and, similarly, can be efficiently inverted with the inverse forward equation

$$\begin{cases} y_{\mathcal{I}_1} &= x_{\mathcal{I}_1} \\ y_{\mathcal{I}_2} &= \big(x_{\mathcal{I}_2} + t_\theta(x_{\mathcal{I}_1})\big) \odot \exp\big(s_\theta(x_{\mathcal{I}_1})\big) \end{cases}$$
$$\Leftrightarrow \begin{cases} x_{\mathcal{I}_1} &= y_{\mathcal{I}_1} \\ x_{\mathcal{I}_2} &= y_{\mathcal{I}_2} \odot \exp\big(-s_\theta(y_{\mathcal{I}_1})\big) - t_\theta(y_{\mathcal{I}_1}). \end{cases}$$

Note that the amount of computation involved in the inversion is no more complex as the forward pass (see figure 5.2b), which is usually already efficient.

Inference As discussed in 5.1.1, computing the log-likelihood of a generator network requires computing its Jacobian determinant. The Jacobian of a coupling layer has the following form

$$\frac{\partial y}{\partial x^T} = \begin{bmatrix} \mathbb{I}_{\mathcal{I}_1} & 0 \\ \frac{\partial y_{\mathcal{I}_2}}{\partial x_{\mathcal{I}_1}^T} & e^{\text{diag}\left(s_\theta(x_{\mathcal{I}_1})\right)} \end{bmatrix},$$

where $e^{\text{diag}\left(s_\theta(x_{\mathcal{I}_1})\right)} = \text{diag}\left(\exp\left(s_\theta(x_{\mathcal{I}_1})\right)\right)$ is a diagonal matrix whose entry are the elements of the vector $\exp\left(s_\theta(x_{\mathcal{I}_1})\right)$. This Jacobian matrix is triangular and, as a result, its determinant can be very efficiently computed as the product of its diagonal

$$\prod_{k=1}^{d_\mathcal{X}} \exp\left(s_\theta(x_{\mathcal{I}_1})\right)_k = \exp\left(\sum_{k=1}^{d_\mathcal{X}} \left(s_\theta(x_{\mathcal{I}_1})\right)_k\right).$$

This shows that the functions s_θ and t_θ affect the complexity of this computation only through their own forward pass, which allow for a large range of expressive models. The triangularity of the unmixing function Jacobian was also exploited in the context of nonlinear independent components analysis in Deco and Brauer (1995a,b), albeit with a fine-grained partitioning similar to autoregressive models.

Gradient flow The architecture of coupling layers bears strong similarities with highway networks (Srivastava et al., 2015), and benefits from similar gradient flow properties, especially its additive variant (Dinh et al., 2014) which is very similar to residual networks (He et al., 2015a, 2016; Gomez et al., 2017) (described subsection 2.2.3).

Composition Although this layer enables several tractable computations of interest and allows the use of powerful models, the associated transformation leaves several components unchanged. Fortunately, this difficulty can be easily overcome by composing coupling layers in an alternating pattern, such that the components

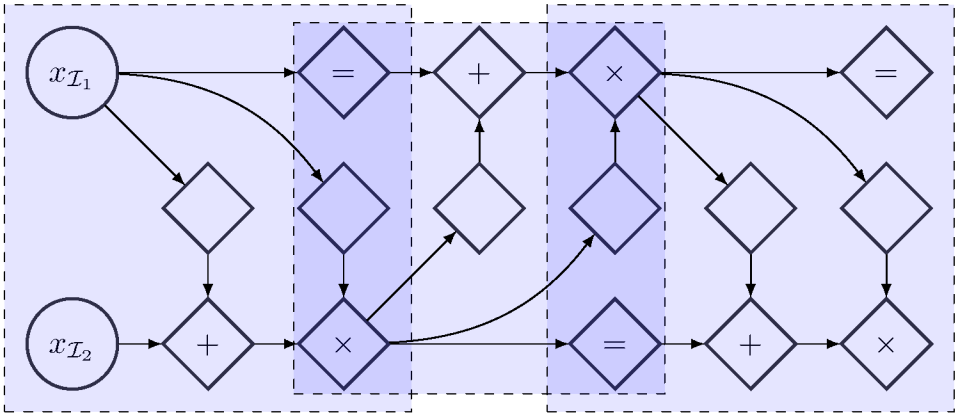


Figure 5.3 – Computational graph of a composition of three coupling layers (each represented by a dashed blue box), from left to right. In this alternating pattern, units which remained identical in one transformation are modified in the next.

that were left unchanged in one coupling layer are updated in the next iteration (see figure 5.3).

This composition conserves the interesting properties of coupling layers as it enables the same efficiency for inversion and sampling,

$$(f_b \circ f_a)^{-1} = f_a^{-1} \circ f_b^{-1},$$

and for computing the Jacobian determinant for log-likelihood tractability,

$$\frac{\partial (f_b \circ f_a)}{\partial x_a^T}(x_a) = \frac{\partial f_b}{\partial x_b^T}\big(f_a(x_a)\big) \cdot \frac{\partial f_a}{\partial x_a^T}(x_a)$$
$$\det(B \cdot A) = \det(B) \cdot \det(A).$$

Observation 1. *Variational autoencoders, as defined in Kingma and Welling (2014a); Rezende et al. (2014), maximize the joint log-likelihood* $\log\big(p_{\theta,\lambda,X,\epsilon}(x,\epsilon)\big)$ *of* (x,ϵ) *where* ϵ *is the auxiliary random variable used for the reparametrization trick (see figure 5.4), with* $-\mathcal{L}_{ELBO}\big(p_{\theta,X,Z}, q_\lambda, x\big) = H(p_\epsilon^*) + \log\big(p_{\theta,\lambda}(x,\epsilon)\big)$. *See figure 5.4.*

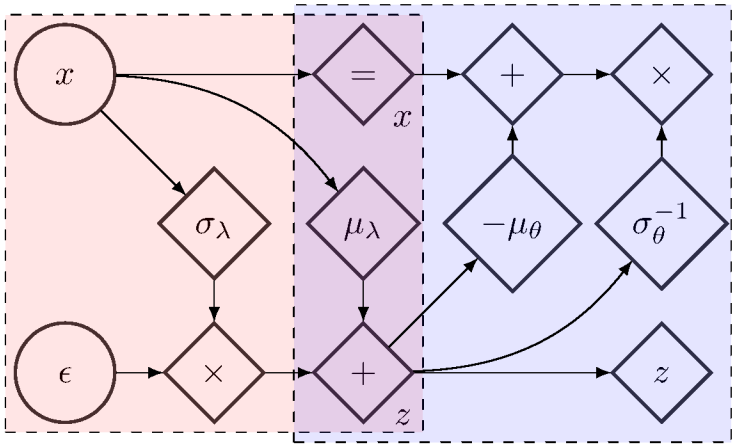


Figure 5.4 – The computational graph of a variational autoencoder is very similar to the composition of two coupling layers, with the encoder in red and the decoder in blue. From that observation, on can conclude that not only variational autoencoders maximize a lower bound on the marginal distribution $\log\big(p_\theta(x)\big)$, they maximize directly a joint distribution $\log\big(p_{\theta,\lambda}(x,\epsilon)\big)$.

Proof. If follows from the derivation

$$
\begin{aligned}
\log\big(p_{\theta,\lambda,X,\epsilon}(x,\epsilon)\big) &= \log\big(p_{\theta,X}(x)p_{\theta,\lambda,\epsilon|X}(\epsilon \mid x)\big) \\
&= \log\left(p_{\theta,X}(x)p_{\theta,Z|X}(z \mid x)\prod_k^{d_{\mathcal{Z}}}\big(\sigma_\lambda(x)\big)_k\right) \\
&= \log\big(p_{\theta,X}(x)p_{\theta,Z|X}(z \mid x)\big) + \sum_k^{d_{\mathcal{Z}}}\log\big(\sigma_\lambda(x)\big)_k \\
&= \log\big(p_{\theta,X,Z}(x,z)\big) + \sum_k^{d_{\mathcal{Z}}}\log\big(\sigma_\lambda(x)\big)_k \\
&= \log\big(p_{\theta,X,Z}(x,z)\big) + \sum_k^{d_{\mathcal{Z}}}\log\big(\sigma_\lambda(x)\big)_k - \log\big(p^*_\epsilon(\epsilon)\big) + \log\big(p^*_\epsilon(\epsilon)\big) \\
&= \log\big(p_{\theta,X,Z}(x,z)\big) - \log\big(q_\lambda(z \mid x)\big) + \log\big(p^*_\epsilon(\epsilon)\big) \\
&= \log\left(\frac{p_{\theta,X,Z}(x,z)}{q_\lambda(z \mid x)}\right) + \log\big(p^*_\epsilon(\epsilon)\big)
\end{aligned}
$$

that

$$
\begin{aligned}
\mathbb{E}_{\epsilon\sim p^*_\epsilon}\Big[\log\big(p_{\theta,\lambda,X,\epsilon}(x,\epsilon)\big)\Big] &= \mathbb{E}_{z\sim q_\lambda(z|x)}\left[\log\left(\frac{p_{\theta,X,Z}(x,z)}{q_\lambda(\cdot \mid x)}\right)\right] + \mathbb{E}_{\epsilon\sim p^*_\epsilon}\Big[\log\big(p^*_\epsilon(\epsilon)\big)\Big] \\
&= -\mathcal{L}_{ELBO}\big(p_{\theta,X,Z}, q_\lambda, x\big) - H(p^*_\epsilon).
\end{aligned}
$$

□

5.2 Preliminary experiments

5.2.1 Setting

General architecture

As a proof of concept, we first experimented with Real NVP with f_θ as a composition of four additive coupling layers with alternating pattern. Each additive coupling layer uses as a translation function t_θ a deep fully-connected rectified neural network with linear output. Because additive coupling layers are actually volume preserving, a final layer, a diagonal matrix parametrized exponentially $\exp(\gamma)$, is appended to this unmixing function. This diagonal matrix can be easily inverted by $\exp(-\gamma)$. The model is trained using the AdaM optimization algorithm (Kingma and Ba, 2015) with parameters $\alpha_t = 10^{-3}$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, a damping coefficient of 10^{-8}, and a batch size of 100.

Datasets

We have trained this model on four image datasets: the *Mixed NIST* dataset (MNIST) (LeCun et al., 1998b), a dataset of 28×28 grayscale images of digits, the *Toronto face dataset* (TFD) (Susskind et al., 2010), a dataset of 48×48 grayscale face images, the *Street View house numbers* (SVHN) (Netzer et al., 2011) dataset, a dataset of 32×32 color images of street numbers, and the *CIFAR-10* dataset (Krizhevsky and Hinton, 2009), a dataset of 32×32 natural color images. For TFD, the unlabeled set is used for training.

Caveats in maximum likelihood learning

The change of variable formula shows us that the scale of the images considered is very important in order to compute the log-likelihood. In our experiments, MNIST, TFD and SVHN are constrained in the $[0, 1]^{d_{\mathcal{X}}}$ hypercube whereas CIFAR-10 is constrained on the $[-1, 1]^d$ hypercube.

Moreover, due to bitmap encoding, images are often initially encoded to be in the quantized space $[\![0, 255]\!]^{d_{\mathcal{X}}}$. It is easy to see that one can obtain an infinitely high test log-likelihood value model by just using a distribution converging to a

Dataset	MNIST	TFD	SVHN	CIFAR-10
Dimensionality $d_{\mathcal{X}}$	$28 \times 28 = 784$	$48 \times 48 = 2304$	$32 \times 32 = 3072$	$32 \times 32 = 3072$
Preprocessing	None	ZCA	ZCA	ZCA
# hidden layers	5	4	4	4
# hidden units	1000	5000	2000	2000
Prior	Logistic	Gaussian	Logistic	Logistic
Mean log-likelihood	1980.50	5514.71	11496.55	5371.78
Bits/dim	4.36	4.55	2.60	4.48

Table 5.1 – Architecture and results for each dataset. # hidden units refer to the number of units per hidden layer.

mixture of Dirac on these $256^{d_{\mathcal{X}}}$ values:

$$p_{\theta,X}(x) = \frac{1}{256^{d_{\mathcal{X}}}} \prod_{i=1}^{d_{\mathcal{X}}} \left(\sum_{k=0}^{255} \delta_k(x_i) \right)$$

The most common to work around this issue is to add a uniform noise $\mathcal{U}([0,1])$ (Uria et al., 2013; van den Oord and Schrauwen, 2014), in order to bridge the quanta, before rescaling to the desired hypercube. It has also the benefit of upper-bounding the log-likelihood of a model by the negentropy of this noise.

Detailed architectures

Whitening (Hyvärinen et al., 2004; Krizhevsky and Hinton, 2009) is used as preprocessing for TFD, SVHN and CIFAR-10. The translation function for MNIST has five hidden layers of 1000 units each, for TFD, we use four layers of 5000 hidden units each, and for CIFAR-10 and SVHN, eah of those four hidden layers have 2000 units. The distribution p_Z is picked to be standard *logistic distribution*

$$p_{Z_k}(z_k) = \sigma_{sigmoid}(z_k)\sigma_{sigmoid}(-z_k)$$

which worked best, except for TFD, where we used a standard Gaussian. The architectures are summarized table 5.1.

5.2.2 Results

Log-likelihood

Table 5.1 shows the quantitative results obtained on these datasets. The mean log-likelihood is computed following the change of variable formula and is averaged over examples. As in Theis and Bethge (2015); van den Oord et al. (2016b), we use also *bits per dimension* (bits/dim), also known as bits per pixel, as a normalized log-likelihood metric. The bits/dim is computed by using the change of variable formula from the mean log-likelihood (averaged over dimensions), with a uniform noise of $\mathcal{U}([0, \delta])$, as follow:

$$\frac{1}{d_{\mathcal{X}} \log(2)} \hat{\mathcal{L}}(p_{\theta,X}, \mathcal{D}_{\text{train}}) - \log_2(\delta).$$

As we mentioned in subsection 5.2.1, as $\hat{\mathcal{L}}(p_{\theta,X}, \mathcal{D}_{\text{train}}) \geq d_{\mathcal{X}} \log(\delta)$, the bits/pixel metric is always positive.

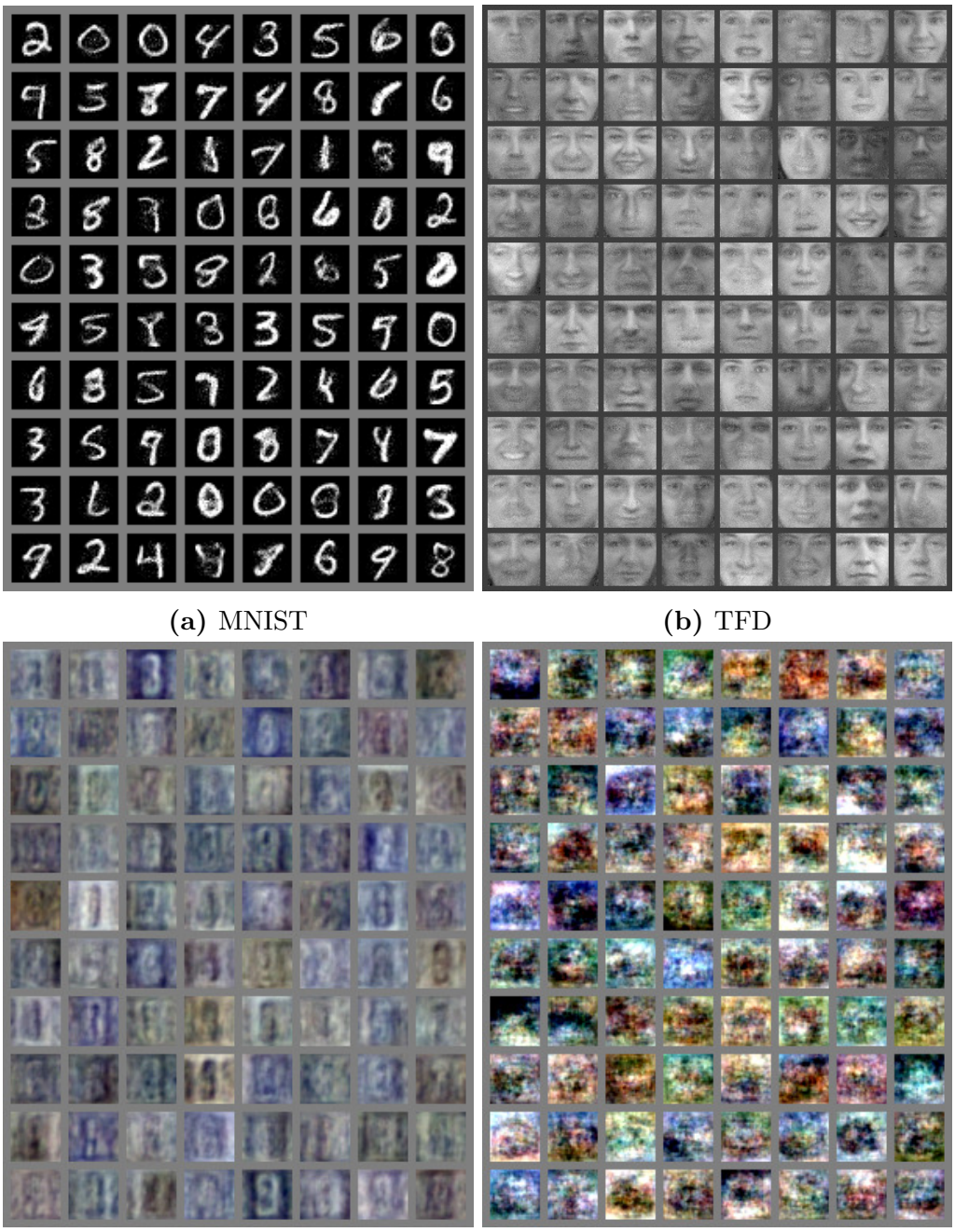
Samples

As mentioned in figure 5.1, we generate unbiased samples x from the resulting density model as follow

$$\begin{aligned} z &\sim p_Z \\ x &= g_\theta(z) = f_\theta^{-1}(z). \end{aligned}$$

Because the support of the modelling density is $\mathbb{R}^{d_{\mathcal{X}}}$ whereas the dataset is actually constrained in a hypercube, we simply choose to clip the resulting samples in this hypercube.

Although the model is able to model reasonably the MNIST and TFD datasets, the architecture fails to capture the complexity of the SVHN and CIFAR-10 datasets (see figure 5.5).



(a) MNIST **(b)** TFD

(c) SVHN **(d)** CIFAR-10

Figure 5.5 – Unbiased samples from a trained model from section 5.2. We sample $z \sim p_Z$ and we output $x = g_\theta(z)$. Although the model is able to model reasonably the MNIST and TFD datasets, the architecture fails to capture the complexity of the SVHN and CIFAR-10 datasets.

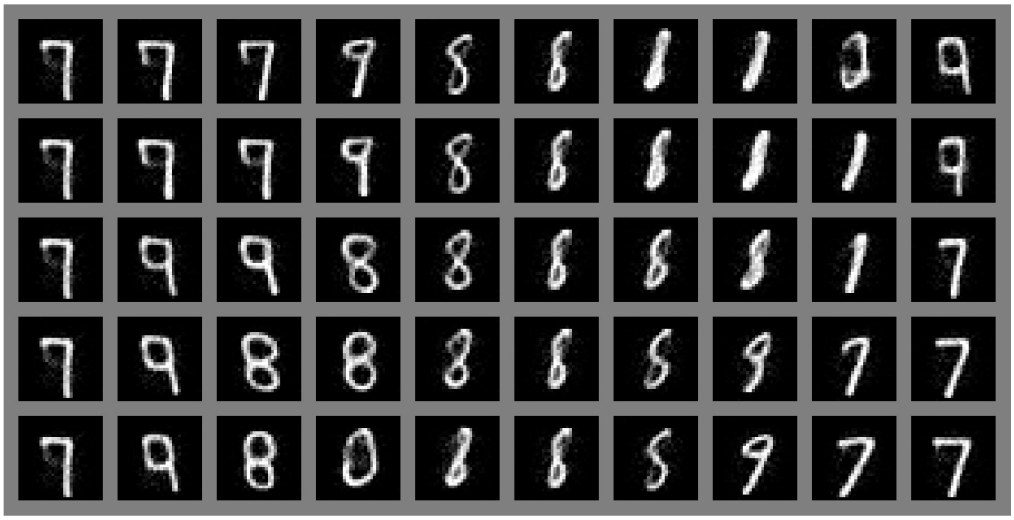


Figure 5.6 – Sphere in the latent space. This figure shows part of the manifold structure learned by the model.

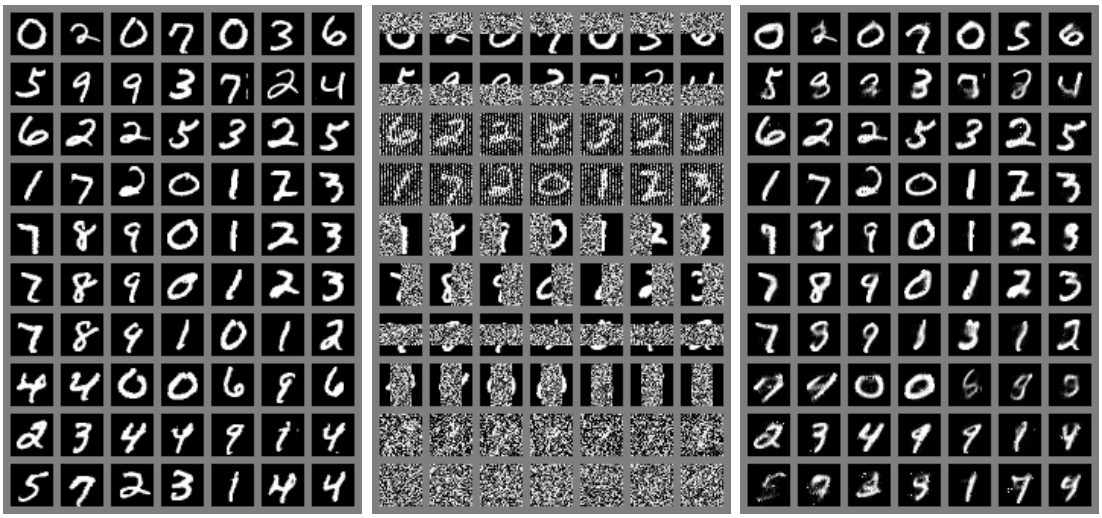
Manifold learning

We can also visualize the latent space learned by the model. We pick a random three dimensional subspace in the latent space and have a unit sphere centered at 0, we sample evenly a grid over this sphere in the angle space and pass the resulting latent vectors through the generator network g_θ to obtain figure 5.6. We see that we can smoothly transition from one digit to another without leaving a plausible manifold of digit images by smoothly transitioning from one latent vector to another.

Inpainting

The model also proved to be interesting in the problem of imputing missing values in an image vector. We chose a partitioning $(\mathcal{I}_{vis}, \mathcal{I}_{hid})$ of visible components and hidden/missing values, we then use the learned model to obtain approximate samples of $x_{\mathcal{I}_{vis}} \mid x_{\mathcal{I}_{hid}}$ by doing an approximate *Langevin sampling* (Welling and Teh, 2011), which can be seen as noisy gradient ascent, on $x_{\mathcal{I}_{vis}}$ using the log-likelihood scoring provided by the model.

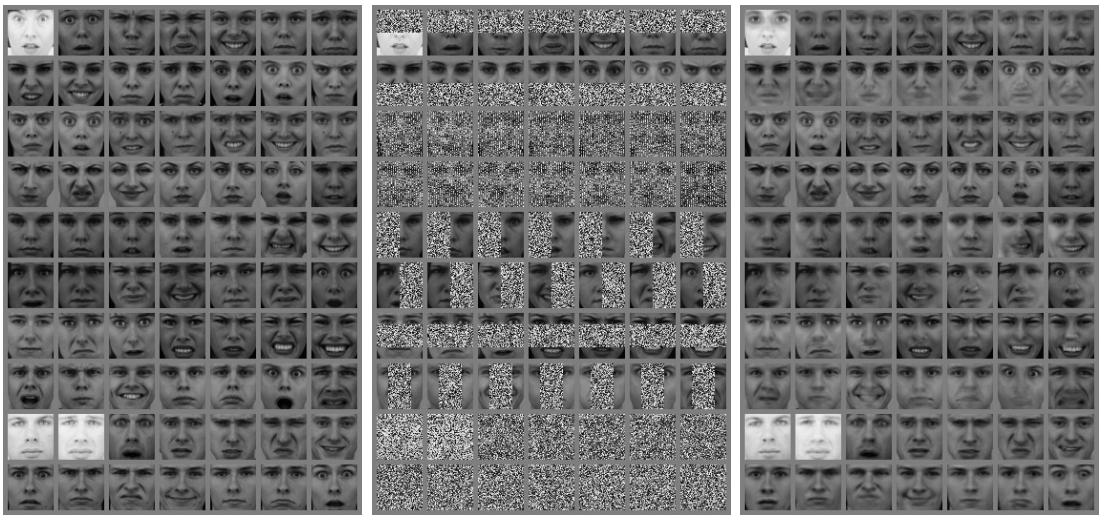
We observe in figure 5.7 that the model does not reconstruct perfectly the original images but is able to infer different original yet plausible images.



(a) Original images for MNIST

(b) Initial states for MNIST

(c) Inference of missing values for MNIST



(d) Original images for TFD

(e) Initial states for TFD

(f) Inference of missing values for TFD

Figure 5.7 – Inpainting experiments. We list below the type of the part of the image masked per line of the above middle figure, from top to bottom: top rows, bottom rows, odd pixels, even pixels, left side, right side, middle vertically, middle horizontally, 75% random, 90% random. We clamp the pixels that are not masked to their ground truth value and infer the state of the masked pixels by *Langevin sampling*. Note that random and odd/even pixels masking are the easiest to solve as neighboring pixels are highly correlated and clamped pixels gives therefore useful information on global structure, whereas more block structured masking results in more difficult inpainting problems.

5.3 Scaling up the model

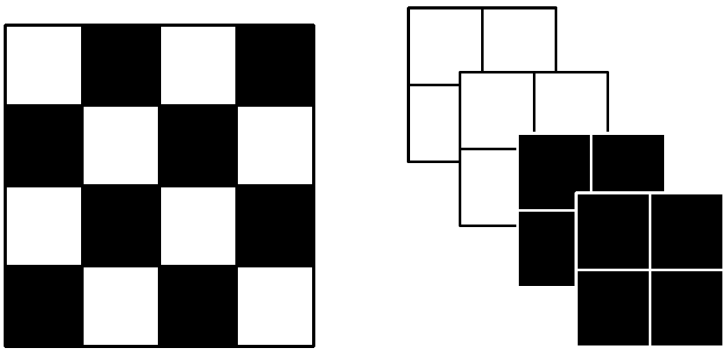
Although we see that this approach is promising, its performance remains insufficient to model accurately larger datasets like SVHN and CIFAR-10. We have hypothesized that this approach would be powerful enough to perform on those tasks and that one main issue in the previous experiments was the training of these coupling layers.

5.3.1 Exploiting image topology

As we will model image datasets to demonstrate the strength of this approach, we will make use of the architectures developed in deep learning applied to computer vision.

Local correlations

In order to exploit image local correlations, we use deep convolutional neural networks (LeCun and Bengio, 1994; Kavukcuoglu et al., 2010; Krizhevsky et al., 2012) as translation and scaling functions for coupling layers. In order to fully harness the use of convolutional networks, we use masking patterns b that are compatible with convolutional structures, for example, that will display some degree of translational invariance and allow local interactions. To that effect, we use *checkerboard masking* and *channel-wise masking* (see figure 5.8).



(a) Checkerboard masking. **(b)** Channel-wise masking.

Figure 5.8 – Examples of convolution compatible masking for coupling layers. Black dots would correspond to the partition $\mathcal{I}_1$ whereas the white ones would correspond to $\mathcal{I}_2$.

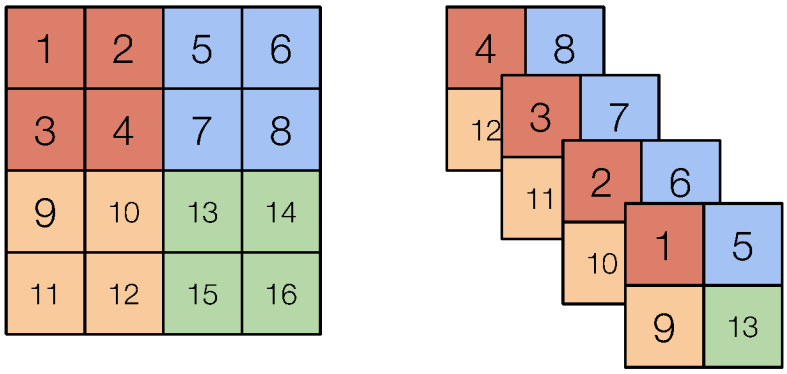


Figure 5.9 – Invertible downsampling operation. We break down the images into 2×2 non-overlapping patches of c channels and convert them into 1×1 patches of $4c$ channels through reshaping. In this figure, $c = 1$.

Global structure

Although considering local structure is very useful when modelling images, natural images cannot be reduced to their local structure. Convolutional architectures for computer vision applications account for this fact by incorporating *pooling* (Zhou and Chellappa, 1988), *strided convolution* (see Dumoulin and Visin, 2016), or *dilated convolution* (Liang-Chieh et al., 2015; Yu and Koltun, 2016; Chen et al., 2016) in their structure. While most of these operations are downsampling operations that results in a loss of invertibility, we will adopt an invertible downsampling scheme as follow: we break down the images into 2×2 non-overlapping patches of c channels and convert them into 1×1 patches of $4 \cdot c$ channels through reshaping (see figure 5.9), this reshapes a $h \times h$ image of c channels into a $\frac{h}{2} \times \frac{h}{2}$ image of $4 \cdot c$ channels. These downsampling operations, resulting in a multi-scale architecture, are very similar to ones involved in strided and dilated convolutions.

5.3.2 Improving gradient flow

Residual connections

As we have noted in subsection 5.1.2 one of the factors that contributed to a successful learning in the preliminary experiments is the strong similarities that coupling layers have with highway networks (Srivastava et al., 2015), and its gradient flow properties, especially its additive variant (Dinh et al., 2014) which is very similar to residual networks (He et al., 2015a, 2016; Gomez et al., 2017) (see subsection 2.2.3). Following van den Oord et al. (2016b), we further exploit this kind

of structure by directly using deep convolutional residual networks with identity connections (He et al., 2016) in our translation and scale networks.

Normalizations

When considering a centered distribution p_Z, one could consider the loss function resulting from the change of variable formula as regression to 0 regularized by the log-Jacobian determinant. As a result, we have observed in our previous experiment a strong signal from gradient descent to have the intermediary representation (output by coupling layers) to be close to 0, which results in poor conditioning and hinders gradient descent progress.

In order to avoid that phenomenon, we use *weight normalization* (Badrinarayanan et al., 2015; Salimans and Kingma, 2016; Arpit et al., 2016) and *batch normalization* (Ioffe and Szegedy, 2015). In particular, we adapt batch normalization to the transformation of probabilistic variables to fit it into the change of variable framework. If we consider a diagonal linear layer

$$z \mapsto a \odot z + b.$$

The associated Jacobian for this layer is simply $\prod_i a_i$. In the context of batch normalization, we estimate online these coefficients as $a = \left(\hat{\sigma}^2_{\hat{\mathcal{M}}} + \nu\right)^{-\frac{1}{2}}$ (with $\nu > 0$ a damping coefficient) and $b = -\hat{\mu}_{\hat{\mathcal{M}}} \odot a$ with the mini-batch $\hat{\mathcal{M}}$ statistics

$$\hat{\mu}_{\hat{\mathcal{M}}} = \frac{1}{\hat{\mathcal{M}}} \sum_{x \in \hat{\mathcal{M}}} x$$
$$\hat{\sigma}^2_{\hat{\mathcal{M}}} = \frac{1}{\hat{\mathcal{M}}} \sum_{x \in \hat{\mathcal{M}}} (x - \hat{\mu}_{\hat{\mathcal{M}}})^2.$$

We use that replacement and therefore use the log-Jacobian determinant

$$-\frac{1}{2} \sum_i \log\left((\hat{\sigma}^2_{\hat{\mathcal{M}}})_i + \nu\right).$$

As in Ioffe and Szegedy (2015), we cumulate the global statistics of training set to obtain the parameters we use at test time.

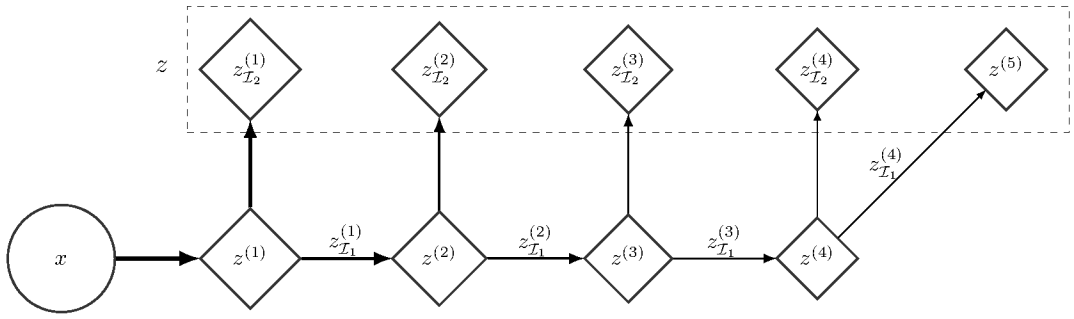


Figure 5.10 – Shortcut connections for factored out components. We can discard components while conserving the the bijectivity of the function if we add shortcut connections from those discarded components to the final output through concatenation.

Shortcuts

Propagating the entire $d_{\mathcal{X}}$ dimensional input through all coupling layers can be expensive, both in terms of computation and memory. Additionally, modelling without further prior can require a large amount of parameters. Fortunately, image classification architectures offer some architectural insights. For instance Simonyan and Zisserman (2015) propose to halve the dimension of the feature space at regular intervals (through pooling). As we can't exactly discard dimensions to keep the computation reversible, we choose instead to stop propagating forward some dimensions and continue transforming instead only the other subset of components. Discarded components are therefore reused later through shortcut connections with concatenation. We factor out the variables as follow (see figure 5.10)

$$
\begin{aligned}
&z^{(1)} = f_\theta^{(1)}(x) && z^{(2)} = f_\theta^{(2)}(z^{(1)}_{\mathcal{I}_1}) && \\
&z^{(3)} = f_\theta^{(3)}(z^{(2)}_{\mathcal{I}_1}) && z^{(4)} = f_\theta^{(4)}(z^{(3)}_{\mathcal{I}_1}) && \\
&\dots && z^{(L)} = f_\theta^{(L)}(z^{(L-1)}_{\mathcal{I}_1}) && z = (z^{(1)}_{\mathcal{I}_2}, \dots, z^{(L-1)}_{\mathcal{I}_2}, z^{(L)}).
\end{aligned}
$$

The model must therefore use less capacity to gaussianize units that were discarded early on, while the architecture allows to use more capacity to disentangle components that are propagated all the way through. This decomposition can be interpreted as multiple layers of gaussianized representation (Salakhutdinov and Hinton, 2009; Rezende et al., 2014; Denton et al., 2015), the further a component is propagated the higher level features it captures, and the units that were discarded early correspond to more local, fine-grained features.

Moreover, gaussianizing and discarding units in earlier layers has the practical

benefit of distributing the loss function throughout the network, following the philosophy similar to guiding intermediate layers using classifiers at several levels (Lee et al., 2014). It also reduces significantly the amount of computation and memory used by the model, allowing us to train larger models.

5.4 Larger-scale experiments

5.4.1 Setting

Bounding the support

As pointed out in section 5.2, the learned density have in general a support in the whole $\mathbb{R}^{d_x}$ space whereas we know that the dataset is constrained to a specific hypercube. For examples, the pixel values of a bitmap image representation typically lie in $[0, 256]^{d_x}$ after application of the recommended jittering procedure (Uria et al., 2013; Theis et al., 2016). Therefore, we instead model the density of $\sigma^{-1}_{sigmoid}\left(\alpha + \frac{(1-\alpha)}{256} \odot x\right)$, where $\alpha \in]0, \frac{1}{2}[$ is constant used in order to reduce the numerical instability stemming from boundary effects. Here we pick $\alpha = 0.05$.

Datasets

We train our model on four natural image datasets: *CIFAR-10* (Krizhevsky and Hinton, 2009) (32×32 images), *small Imagenet* (Russakovsky et al., 2015; van den Oord et al., 2016b), *Large-scale Scene Understanding (LSUN)* (Yu et al., 2015), *CelebFaces Attributes (CelebA)* (Liu et al., 2015). More specifically, we train on the downsampled to 32×32 and 64×64 versions of Imagenet (van den Oord et al., 2016b). For the LSUN dataset, we train on the *bedroom*, *tower* and *church (outdoor)* categories. The procedure for LSUN is the same as in Radford et al. (2015): we downsample the image so that the smallest side is 96 pixels and take random uniform crops of 64×64. For CelebA, we use the same procedure as in (Larsen et al., 2015b): we take an approximately central crop of 148×148 then resize it to 64×64.

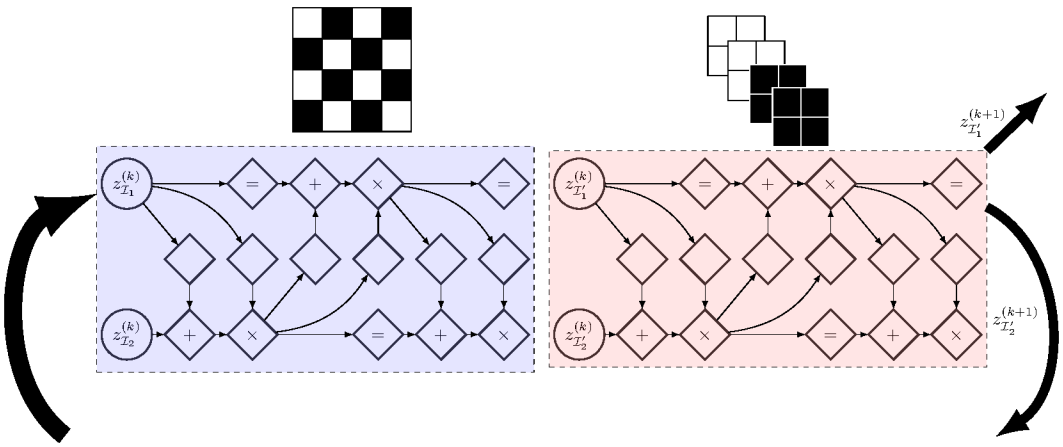


Figure 5.11 – Multi-scale architecture. We recursively build this architecture by applying three affine coupling layers with checkerboard masking before performing invertible downsampling and applying three affine coupling layers with channel-wise masking. We discard half of the channels of the output (as in figure 5.10). We then repeat this operation until the result is 4×4.

Architecture

We detail the architecture we proposed in subsection 5.3.2. For $f_\theta^{(l)}$ with $l < L - 1$, we use three affine coupling layers with checkerboard masking composed in an alternating pattern, then apply an invertible downsampling described in subsection 5.3.1 before finally applying three other affine coupling layers with channel-wise masking. We then discard half of the channels of the resulting output. For $f^{(L)}$, we simply use four affine coupling layers with checkerboard masking. This multi-scale architecture is applied until the final output is of size 4×4 (see figure 5.11), e.g. a 32×32 image datasets will set $L = 5$. The damping coefficient for batch normalization is set to $\nu = 10^{-6}$. p_Z is set to $\mathcal{N}(0, \mathbb{I}_{d_\mathcal{X}})$

The function that outputs $(s_\theta(b \odot x), t_\theta(b \odot x))$ for each affine coupling layer is a deep rectified convolutional residual network with identity residual connections (He et al., 2016) and linear output. If we consider this function as $\text{NN}\text{ET}_\theta(b \odot x)$ then the transformation quantities are computed as

$$
\begin{aligned}
t_\theta(b \odot x) &= \text{NN}\text{ET}_\theta(b \odot x)_{\mathcal{I}_t} \\
s_\theta(b \odot x) &= \gamma \phi_{tanh}\big(\text{NN}\text{ET}_\theta(b \odot x)_{\mathcal{I}_s}\big),
\end{aligned}
$$

with γ a scalar learned parameter. For 32×32 image size dataset, we use for the

Dataset	PixelRNN (van den Oord et al., 2016b)	Real NVP	Conv DRAW (Gregor et al., 2016)	IAF-VAE (Kingma et al., 2016)
CIFAR-10	3.00	3.49	< 3.59	< 3.11
Imagenet (32×32)	3.86 (3.83)	4.28 (4.26)	< 4.40 (4.35)	
Imagenet (64×64)	3.63 (3.57)	3.98 (3.75)	< 4.10 (4.04)	
LSUN (bedroom)		2.72 (2.70)		
LSUN (tower)		2.81 (2.78)		
LSUN (church outdoor)		3.08 (2.94)		
CelebA		3.02 (2.97)		

Table 5.2 – Bits/dim results for CIFAR-10, Imagenet, LSUN datasets and CelebA. Test results for CIFAR-10 and validation results for Imagenet, LSUN and CelebA (with training results in parenthesis for reference).

first three coupling layers a residual network with four residual blocks, with one hidden layer, with every hidden layer containing 32 hidden feature maps. Only two residual blocks are used for images of size 64×64. For CIFAR-10, we use eight residual blocks, 64 feature maps, and downscale only once. After each downscaling, we double the number of hidden feature maps used. We use the AdaM optimizer with the same parameters as section 5.2, except with a batch size of 64. We also add a small L_2 regularizer $\|\theta\|_2^2$ on the weight scale parameters (in weight normalization parametrization) with a coefficient of $5 \cdot 10^{-5}$. The images are randomly flipped horizontally.

5.4.2 Results

Log-likelihood

We show in table 5.2 that, while not improving over more modern methods like *pixel recurrent neural network* (*PixelRNN*) (van den Oord et al., 2016b) and variational autoencoders with *inverse autoregressive flow* (*IAF*) (Kingma et al., 2016), the number of bits per dimension obtained is competitive with other generative methods. As we notice that our performance increases with the number of parameters, we hypothesize that we are in an underfitting regime where larger models are likely to further improve performances. For CelebA and LSUN, the number of bits per dimension for the validation set was decreasing throughout training, so little overfitting is expected.

Figure 5.12 – On the left column, examples from the dataset. On the right column, samples from the model trained on the dataset. The datasets shown in this figure are in order: CIFAR-10, Imagenet (32×32), Imagenet (64×64), CelebA, LSUN (bedroom).

Samples

We show in figure 5.12 samples generated from the model with training examples from the dataset for comparison. As mentioned in (Theis et al., 2016; Gregor et al., 2016), maximum likelihood is a principle that values diversity over sample quality in a limited capacity setting. As a result, our model outputs sometimes highly improbable samples as we can notice especially on CelebA. As opposed to variational autoencoders with diagonal Gaussian approximate posterior, the samples generated from our model look not only globally coherent but also sharp. A hypothesis is that as opposed to these models, Real NVP does not rely on fixed form reconstruction cost like an L_2 norm which tends to reward capturing low frequency components more heavily than high frequency components. The multi-

scale architecture used is also able to effectively capture the global structure of these image datasets. Unlike autoregressive models, sampling from our model is done very efficiently as it is parallelized over input dimensions. On Imagenet and LSUN, our model seems to have captured well the notion of background/foreground and lighting interactions such as luminosity and consistent light source direction for reflectance and shadows.

(a) Imagenet (64×64)

(b) CelebA

(c) LSUN (bedroom)

(d) LSUN (church outdoor)

Figure 5.13 – We exploit the convolutional structure of our generative model to generate samples $\times 10$ bigger than the training set image size. The model perform best on random crops generated datasets like LSUN categories.

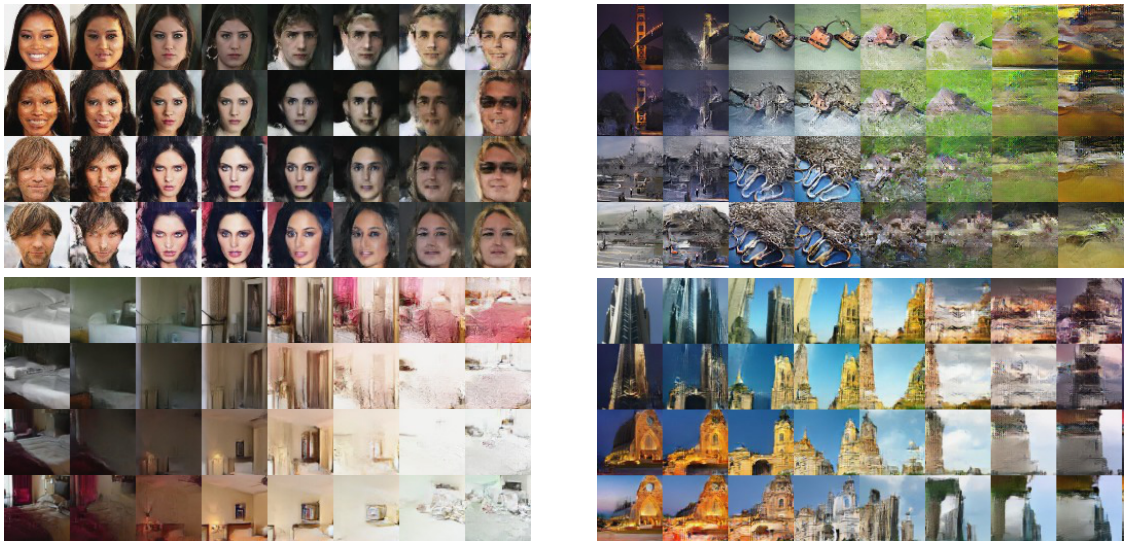


Figure 5.14 – Manifold generated from four examples in the dataset. Clockwise from top left: CelebA, Imagenet (64×64), LSUN (tower), LSUN (bedroom).

Inspired by the texture generation work by (Gatys et al., 2015; Theis and Bethge, 2015) and extrapolation test with DCGAN (Radford et al., 2015), we also evaluate the statistics captured by our model by generating images ten times as large as present in the dataset (see figure 5.13). As we can observe in the following figures, our model seems to successfully create a "texture" representation of the dataset while maintaining a spatial smoothness through the image. Our convolutional architecture is only aware of the position of considered pixels through edge effects in convolutions, which makes our model well suited to model a stationary process. This also explains why these samples are more coherent with the training distribution in LSUN, where the training data was obtained using random crops.

Manifold learning

We also illustrate the smooth semantically consistent meaning of our latent variables. In the latent space, we define a manifold based on four validation examples $z_{(1)}$, $z_{(2)}$, $z_{(3)}$, $z_{(4)}$, and parametrized by two parameters ϕ and ϕ' by,

$$z = \cos(\phi)\left(\cos(\phi')z_{(1)} + \sin(\phi')z_{(2)}\right) + \sin(\phi)\left(\cos(\phi')z_{(3)} + \sin(\phi')z_{(4)}\right).$$

We project the resulting manifold back into the data space by computing $g(z)$. This resulting manifold is very similar to a sphere in latent space and points sampled uniformly in angle space are shown figure 5.14. We observe that the model seems to have organized the latent space such that a smooth transition between latent vectors in $\mathcal{Z}$ results in a smooth transition in $\mathcal{X}$ while keeping plausible images during the transition, unlike mere pixel space interpolation. This reasoning faces of course its limit when transition between two completely unrelated examples, bringing the dilemma of either forcing sharp transition or leaving implausible samples on the path.

Structure of the latent space

As in (Gregor et al., 2016), we further try to grasp the semantic of our learned layers latent variables by doing ablation tests. We infer the latent variables and resample the lowest levels of latent variables from a standard Gaussian, increasing the highest level affected by this resampling (see figure 5.15). As we can see in figure 5.16, the semantic of our latent space seems to be more on a graphic level rather than higher level concept. Although the heavy use of convolution improves learning by exploiting image prior knowledge, it is also likely to be responsible for this limitation.

Let's rewrite an example of the equation for the shortcut connections enabling the discarding of components at different levels for 64×64 images model:

$$
\begin{aligned}
z^{(1)} &= f_\theta^{(1)}(x) & z^{(2)} &= f_\theta^{(2)}(z_{\mathcal{I}_1}^{(1)}) \\
z^{(3)} &= f_\theta^{(3)}(z_{\mathcal{I}_1}^{(2)}) & z^{(4)} &= f_\theta^{(4)}(z_{\mathcal{I}_1}^{(3)}) \\
z^{(5)} &= f_\theta^{(5)}(z_{\mathcal{I}_1}^{(4)}) & z &= (z_{\mathcal{I}_2}^{(1)}, z_{\mathcal{I}_2}^{(2)}, z_{\mathcal{I}_2}^{(3)}, z_{\mathcal{I}_2}^{(4)}, z^{(5)}).
\end{aligned}
$$

If we resample using a standard Gaussian ϵ then we would display in figure 5.16

$$
\begin{aligned}
g_\theta\big((z_{\mathcal{I}_2}^{(1)}, z_{\mathcal{I}_2}^{(2)}, z_{\mathcal{I}_2}^{(3)}, z_{\mathcal{I}_2}^{(4)}, z^{(5)})\big) &\qquad \textit{(original image)} \\
g_\theta\big((\epsilon_{\mathcal{I}_2}^{(1)}, z_{\mathcal{I}_2}^{(2)}, z_{\mathcal{I}_2}^{(3)}, z_{\mathcal{I}_2}^{(4)}, z^{(5)})\big) &\qquad \textit{(minimal corruption)} \\
\dots & \\
g_\theta\big((\epsilon_{\mathcal{I}_2}^{(1)}, \epsilon_{\mathcal{I}_2}^{(2)}, \epsilon_{\mathcal{I}_2}^{(3)}, \epsilon_{\mathcal{I}_2}^{(4)}, z^{(5)})\big) &\qquad \textit{(maximal corruption).}
\end{aligned}
$$

$g_\theta\big((\epsilon_{\mathcal{I}_2}^{(1)}, \epsilon_{\mathcal{I}_2}^{(2)}, \epsilon_{\mathcal{I}_2}^{(3)}, \epsilon_{\mathcal{I}_2}^{(4)}, \epsilon^{(5)})\big)$ would then correspond to a complete resampling.

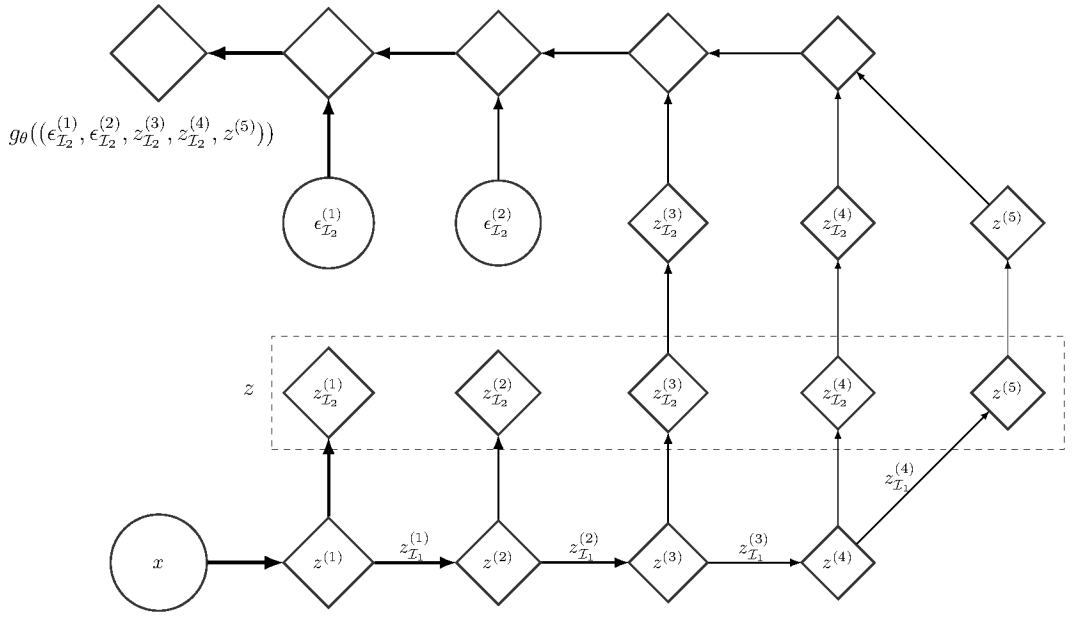


Figure 5.15 – Ablation of the latent variable by resampling components. Here we represent the computational graph for generating $g_\theta\big((\epsilon^{(1)}_{\mathcal{I}_2}, \epsilon^{(2)}_{\mathcal{I}_2}, z^{(3)}_{\mathcal{I}_2}, z^{(4)}_{\mathcal{I}_2}, z^{(5)})\big)$.

In this particular decomposition, we can see that $z^{(1)}_{\mathcal{I}_2}$ corresponds to simpler, yet fine-grained, features while $z^{(5)}$ corresponds on the contrary to more complex, global components. As opposed to traditional emission models using simple isotropic noise to enable log-likelihood learning, the noise model corresponding to $z^{(1)}_{\mathcal{I}_2}$ allows for more structured noise that would better correspond to local correlation structure of images, and yet, would result in a still small overhead as $z^{(1)}_{\mathcal{I}_2}$ is discarded early on in the disentangling process. This is one of the possible ways one could lighten the burden of constraining the model to be bijective: instead of summarizing the essential information of the dataset into a lower dimensional latent space, we use a latent space with identical dimensionality but organized from global complex features to simple fine-grained features, which does not necessarily require much more capacity, both in terms of expressivity and computation, than other approaches and allows to select after training the components one finds most relevant.

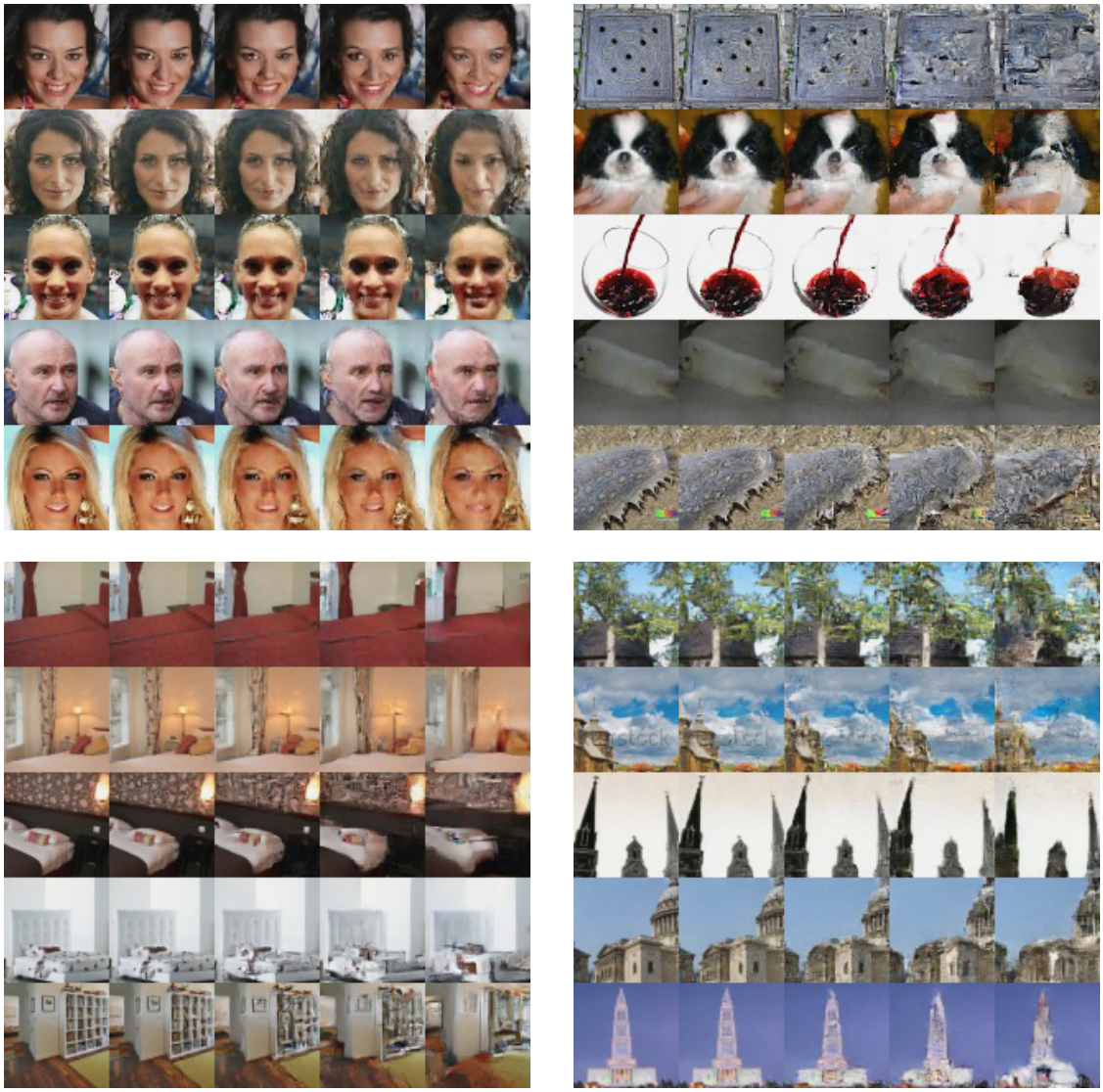


Figure 5.16 – Conceptual compression from a model trained on CelebA (top left), Imagenet (64×64) (top right), LSUN (bedroom) (bottom left), and LSUN (church outdoor) (bottom right). The leftmost column represent the original image, the subsequent columns were obtained by storing higher level latent variables and resampling the others, storing less and less as we go right. From left to right: 100%, 50%, 25%, 12.5%, and 6.25% of the latent variables are kept.

5.5 Discussion

In this chapter, we have defined a class of invertible functions with tractable Jacobian determinant, enabling exact and tractable log-likelihood evaluation, inference, and sampling. We have shown that this class of generative models achieves competitive performance, both in terms of sample quality and log-likelihood.

Although, the model enabled efficient and exact log-likelihood closed form evaluation and equally efficient sampling, the drawback that was brought by architectural constraints proved to be a sufficient hurdle to slightly underperform on the log-likelihood metric (which proved to be more valued when available than sampling efficiency) compared to more modern techniques like *pixel recurrent/convolutional neural network* (*PixelRNN/CNN*) (van den Oord et al., 2016b,a) and variational autoencoders with *inverse autoregressive flow* (*IAF*) (Kingma et al., 2016). As Rainforth et al. (2017) demonstrated how tighter variational bounds does not necessarily improve associated models, we can wonder to which extent optimizing the log-likelihood directly might result in a problem harder than a more relaxed evidence lower bound optimization problem. Moreover, although the variational autoencoder can only maximize an evidence lower bound during training, this class of models have more flexibility in the class of generator network and inference function it can use. For example, Rezende and Mohamed (2015); Kingma et al. (2016) harnesses the ability fo variational autoencoders to use flows with inverse that are expensive or lacking closed form expression. At this stage, it remains unknown whether this lack of effective capacity comes from a lack of expressivity or optimization difficulties, which leaves this venue open for further research.

Although this work missed the objective of building a state-of-the-art generative models, the insights brought here and the techniques developed proved to be useful in the domain of deep probabilistic modelling:

- it contributed to popularize the change of variable formula in deep learning which was also useful for stochastic variational inference through the reparametrization trick, as described in the seminal work on *variational inference through normalizing flows* (Rezende and Mohamed, 2015; Tomczak and Welling, 2016, 2017; Louizos and Welling, 2017; Kingma et al., 2016) which contributed to overcome a significant hurdle in training powerful variational autoencoder models;

- it reintroduced the formulation of real-valued autoregressive models as continuous bijective transformations with triangular Jacobian (Deco and Brauer, 1995a,b; Hyvärinen and Pajunen, 1999) in modern deep learning, which proved to be useful in the conception of deep probabilistic models (Kingma et al., 2016; Papamakarios et al., 2017; van den Oord et al., 2017);

- the trainable invertible architecture from the layering of coupling layers enabled the design of trainable *reversible Markov chain* to learn more efficient *Markov chain Monte Carlo* algorithms (Song et al., 2017; Levy et al., 2017).

Moreover, this same trainable invertible architecture was crucial in the design of more memory-efficient backpropagation through *reversible neural networks* architectures (Gomez et al., 2017), by efficiently recomputing necessary quantities instead of storing them in memories. This type of architecture with invertible components enables the design of experiments (Jacobsen et al., 2018) that have recently cast doubt on the conventional wisdom of the necessity of discarding information in feature space in order to obtain performing classifiers.

Conclusion

The work presented in this thesis aimed at deepening our understanding of the three essential questions for the design of deep learning architectures described in chapter 1: expressivity, trainability, and generalizability. In particular, this work approached these themes in the context of *deep learning*, described in chapter 2, through the lens of reparametrization in order to confront several important research directions.

Chapter 3 demonstrates how the notion of *flatness* in the loss function, which is equivariant to reparametrization and the incurred change in loss function geometry, cannot be directly correlated with a low *generalization gap*, as it is invariant to reparametrization and loss function geometry, even for popular deep learning architectures. As a result, this work challenged an otherwise widespread belief on sharp/flat minima resulting in high/low generalization gap.

In chapter 5, we explore the *change of variable formula*, which quantifies the density resulting from bijective reparametrizations of the input space. We designed Real NVP, an expressive architecture trainable on this closed form expression which proved to be competitive for generative modelling (which we described in chapter 4), in an attempt to question the necessity of an approximation scheme adopted for learning generator networks. Instead, it resulted in useful insights and techniques in the development of these approximation schemes.

This work have attemped to tackle central deep learning problems which calls upon new research avenues. For example, more promising analyses on generalization performance of deep models would be likely to include a holistic view of our algorithms, including loss function and stochastic optimization procedure altogether. The failure of Real NVP in providing state-of-the-art generative models brings new incentive in designing more efficient models with similar properties, by relaxing the bijectivity constraint or using discrete variables.

Bibliography

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., et al. (2016). Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*.

Ahn, S., Balan, A. K., and Welling, M. (2012). Bayesian posterior sampling via stochastic gradient fisher scoring. In Langford and Pineau (2012).

Aksoy, S. and Haralick, R. M. (2001). Feature normalization and likelihood-based similarity measures for image retrieval. *Pattern recognition letters*, 22(5):563–582.

Ali, S. M. and Silvey, S. D. (1966). A general class of coefficients of divergence of one distribution from another. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 131–142.

Amari, S.-I. (1998). Natural gradient works efficiently in learning. *Neural Comput.*, 10(2).

Andrieu, C., De Freitas, N., Doucet, A., and Jordan, M. I. (2003). An introduction to mcmc for machine learning. *Machine learning*, 50(1-2):5–43.

Arpit, D., Zhou, Y., Kota, B. U., and Govindaraju, V. (2016). Normalization propagation: A parametric technique for removing internal covariate shift in deep networks. *arXiv preprint arXiv:1603.01431*.

Bach, F. R. and Blei, D. M., editors (2015). *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, volume 37 of *JMLR Workshop and Conference Proceedings*. JMLR.org.

Badrinarayanan, V., Mishra, B., and Cipolla, R. (2015). Understanding symmetries in deep networks. *arXiv preprint arXiv:1511.01029*.

Bahdanau, D., Cho, K., and Bengio, Y. (2015). Neural machine translation by jointly learning to align and translate. In *ICLR'2015, arXiv:1409.0473*.

Baird, L., Smalenberger, D., and Ingkiriwang, S. (2005). One-step neural network inversion with pdf learning and emulation. In *Neural Networks, 2005. IJCNN'05. Proceedings. 2005 IEEE International Joint Conference on*, volume 2, pages 966–971. IEEE.

Balcan, M. and Weinberger, K. Q., editors (2016). *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, volume 48 of *JMLR Workshop and Conference Proceedings*. JMLR.org.

Balduzzi, D., Frean, M., Leary, L., Lewis, J. P., Ma, K. W., and McWilliams, B. (2017). The shattered gradients problem: If resnets are the answer, then what is the question? In Precup and Teh (2017), pages 342–350.

Barron, A. R. (1993). Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Transactions on Information theory*, 39(3):930–945.

Bartlett, P. L. and Mendelson, S. (2002). Rademacher and gaussian complexities: Risk bounds and structural results. *Journal of Machine Learning Research*, 3(Nov):463–482.

Bastien, F., Lamblin, P., Pascanu, R., Bergstra, J., Goodfellow, I., Bergeron, A., Bouchard, N., Warde-Farley, D., and Bengio, Y. (2012). Theano: new features and speed improvements. *arXiv preprint arXiv:1211.5590*.

Beck, A. and Teboulle, M. (2003). Mirror descent and nonlinear projected subgradient methods for convex optimization. *Operations Research Letters*, 31(3):167–175.

Bengio, Y. (1991). Artificial neural networks and their application to sequence recognition.

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

Bengio, Y. and Bengio, S. (1999). Modeling high-dimensional discrete data with multi-layer neural networks. In *NIPS*, volume 99, pages 400–406.

Bengio, Y. and Delalleau, O. (2011). On the expressive power of deep architectures. In Kivinen, J., Szepesvári, C., Ukkonen, E., and Zeugmann, T., editors, *Algorithmic Learning Theory - 22nd International Conference, ALT 2011, Espoo, Finland, October 5-7, 2011. Proceedings*, volume 6925 of *Lecture Notes in Computer Science*, pages 18–36. Springer.

Bengio, Y., Delalleau, O., and Roux, N. L. (2006). The curse of highly variable functions for local kernel machines. In *Advances in neural information processing systems*, pages 107–114.

Bengio, Y., Ducharme, R., Vincent, P., and Janvin, C. (2003). A neural probabilistic language model. *Journal of Machine Learning Research*, 3:1137–1155.

Bengio, Y., Laufer, E., Alain, G., and Yosinski, J. (2014). Deep generative stochastic networks trainable by backprop. In Xing and Jebara (2014), pages 226–234.

Bengio, Y., LeCun, Y., et al. (2007). Scaling learning algorithms towards ai. *Large-scale kernel machines*, 34(5):1–41.

Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166.

Bergstra, J., Breuleux, O., Bastien, F., Lamblin, P., Pascanu, R., Desjardins, G., Turian, J., Warde-Farley, D., and Bengio, Y. (2010). Theano: a cpu and gpu math expression compiler. In *Proceedings of the Python for scientific computing conference (SciPy)*, volume 4, page 3. Austin, TX.

Bishop, C. M. (1993). Curvature-driven smoothing: a learning algorithm for feedforward networks. *IEEE Transactions on Neural Networks*, 4(5):882–884.

Bishop, C. M. (2006). *Pattern recognition and machine learning*. springer.

Bottou, L. (1998). Online learning and stochastic approximations. *On-line learning in neural networks*, 17(9):142.

Bottou, L. and Bousquet, O. (2008). The tradeoffs of large scale learning. In Platt, J., Koller, D., Singer, Y., and Roweis, S., editors, *Advances in Neural Information Processing Systems*, volume 20, pages 161–168. NIPS Foundation (http://books.nips.cc).

Bottou, L., Curtis, F. E., and Nocedal, J. (2016). Optimization methods for large-scale machine learning. *arXiv preprint arXiv:1606.04838*.

Bousquet, O. and Elisseeff, A. (2002). Stability and generalization. *Journal of Machine Learning Research*, 2(Mar):499–526.

Breiman, L. (1996). Bagging predictors. *Machine learning*, 24(2):123–140.

Burda, Y., Grosse, R., and Salakhutdinov, R. (2016). Importance weighted autoencoders. In *ICLR'2016, arXiv:1509.00519*.

Chan, W., Jaitly, N., Le, Q. V., and Vinyals, O. (2016). Listen, attend and spell: A neural network for large vocabulary conversational speech recognition. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2016, Shanghai, China, March 20-25, 2016*, pages 4960–4964. IEEE.

Chaudhari, P., Choromanska, A., Soatto, S., LeCun, Y., Baldassi, C., Borgs, C., Chayes, J., Sagun, L., and Zecchina, R. (2017). Entropy-sgd: Biasing gradient descent into wide valleys. In *ICLR'2017, arXiv:1611.01838*.

Chen, L.-C., Papandreou, G., Kokkinos, I., Murphy, K., and Yuille, A. L. (2016). Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *arXiv preprint arXiv:1606.00915*.

Chen, S. S. and Gopinath, R. A. (2001). Gaussianization. In Leen, T. K., Dietterich, T. G., and Tresp, V., editors, *Advances in Neural Information Processing Systems 13*, pages 423–429. MIT Press.

Cho, K., van Merrienboer, B., Gülçehre, Ç., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. In Moschitti, A., Pang, B., and Daelemans, W., editors, *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar,*

A meeting of SIGDAT, a Special Interest Group of the ACL, pages 1724–1734. ACL.

Choromanska, A., Henaff, M., Mathieu, M., Arous, G. B., and LeCun, Y. (2015). The loss surfaces of multilayer networks. In *AISTATS*.

Chorowski, J. K., Bahdanau, D., Serdyuk, D., Cho, K., and Bengio, Y. (2015). Attention-based models for speech recognition. In *Advances in Neural Information Processing Systems*, pages 577–585.

Coates, A., Ng, A., and Lee, H. (2011). An analysis of single-layer networks in unsupervised feature learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 215–223.

Collobert, R., Puhrsch, C., and Synnaeve, G. (2016). Wav2letter: an end-to-end convnet-based speech recognition system. *arXiv preprint arXiv:1609.03193*.

Csiszár, I., Shields, P. C., et al. (2004). Information theory and statistics: A tutorial. *Foundations and Trends® in Communications and Information Theory*, 1(4):417–528.

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems (MCSS)*, 2(4):303–314.

Dahl, G. E., Yu, D., Deng, L., and Acero, A. (2012). Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition. *IEEE Transactions on audio, speech, and language processing*, 20(1):30–42.

Dauphin, Y., de Vries, H., and Bengio, Y. (2015). Equilibrated adaptive learning rates for non-convex optimization. In *Advances in neural information processing systems*, pages 1504–1512.

Dauphin, Y. N., Pascanu, R., Gülçehre, Ç., Cho, K., Ganguli, S., and Bengio, Y. (2014). Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. *NIPS*.

Dayan, P., Hinton, G. E., Neal, R. M., and Zemel, R. S. (1995). The helmholtz machine. *Neural computation*, 7(5):889–904.

Deco, G. and Brauer, W. (1995a). Higher order statistical decorrelation without information loss. In *Advances in Neural Information Processing Systems*, pages 247–254.

Deco, G. and Brauer, W. (1995b). Nonlinear higher-order statistical decorrelation by volume-conserving neural architectures. *Neural Networks*, 8(4):525–535.

Denton, E., Chintala, S., Szlam, A., and Fergus, R. (2015). Deep generative image models using a laplacian pyramid of adversarial networks. *arXiv preprint arXiv:1506.05751*.

Desjardins, G., Simonyan, K., Pascanu, R., et al. (2015). Natural neural networks. In *Advances in Neural Information Processing Systems*, pages 2071–2079.

Devroye, L. (1986). Sample-based non-uniform random variate generation. In *Proceedings of the 18th conference on Winter simulation*, pages 260–265. ACM.

Dinh, L., Krueger, D., and Bengio, Y. (2014). Nice: Non-linear independent components estimation. *arXiv preprint arXiv:1410.8516*.

Dinh, L., Pascanu, R., Bengio, S., and Bengio, Y. (2017). Sharp minima can generalize for deep nets. In Precup and Teh (2017), pages 1019–1028.

Dinh, L., Sohl-Dickstein, J., and Bengio, S. (2016). Density estimation using real nvp. In *ICLR'2017, arXiv:1605.08803*.

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

Dwork, C., Feldman, V., Hardt, M., Pitassi, T., Reingold, O., and Roth, A. (2015). The reusable holdout: Preserving validity in adaptive data analysis. *Science*, 349(6248):636–638.

Dziugaite, G. K. and Roy, D. M. (2017). Computing nonvacuous generalization bounds for deep (stochastic) neural networks with many more parameters than training data. In Elidan, G., Kersting, K., and Ihler, A. T., editors, *Proceedings of the Thirty-Third Conference on Uncertainty in Artificial Intelligence, UAI 2017, Sydney, Australia, August 11-15, 2017*. AUAI Press.

Eldan, R. and Shamir, O. (2016). The power of depth for feedforward neural networks. In *Conference on Learning Theory*, pages 907–940.

Ermoliev, Y. (1983). Stochastic quasigradient methods and their application to system optimization. *Stochastics: An International Journal of Probability and Stochastic Processes*, 9(1-2):1–36.

Frey, B. J. (1998). *Graphical models for machine learning and digital communication*. MIT press.

Gal, Y. (2016). *Uncertainty in deep learning*. PhD thesis, PhD thesis, University of Cambridge.

Gatys, L. A., Ecker, A. S., and Bethge, M. (2015). Texture synthesis using convolutional neural networks. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 262–270.

Gehring, J., Auli, M., Grangier, D., and Dauphin, Y. N. (2016). A convolutional encoder model for neural machine translation. *arXiv preprint arXiv:1611.02344*.

Germain, M., Gregor, K., Murray, I., and Larochelle, H. (2015). MADE: masked autoencoder for distribution estimation. *CoRR*, abs/1502.03509.

Glorot, X., Bordes, A., and Bengio, Y. (2011). Deep sparse rectifier neural networks. In *Aistats*, volume 15, page 275.

Glynn, P. W. (1990). Likelihood ratio gradient estimation for stochastic systems. *Commun. ACM*, 33(10):75–84.

Gomez, A. N., Ren, M., Urtasun, R., and Grosse, R. B. (2017). The reversible residual network: Backpropagation without storing activations. In *Advances in Neural Information Processing Systems*, pages 2211–2221.

Gonen, A. and Shalev-Shwartz, S. (2017). Fast rates for empirical risk minimization of strict saddle problems. *arXiv preprint arXiv:1701.04271*.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep learning*. MIT press.

Goodfellow, I. J., Bulatov, Y., Ibarz, J., Arnoud, S., and Shet, V. (2014a). Multi-digit number recognition from street view imagery using deep convolutional neural networks. In *ICLR'2014, arXiv:1312.6082*.

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A. C., and Bengio, Y. (2014b). Generative adversarial nets. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 2672–2680.

Goodfellow, I. J., Shlens, J., and Szegedy, C. (2015). Explaining and harnessing adversarial examples. In *ICLR'2015 arXiv:1412.6572*.

Goodfellow, I. J., Warde-Farley, D., Mirza, M., Courville, A. C., and Bengio, Y. (2013). Maxout networks. *ICML (3)*, 28:1319–1327.

Goyal, P., Dollár, P., Girshick, R., Noordhuis, P., Wesolowski, L., Kyrola, A., Tulloch, A., Jia, Y., and He, K. (2017). Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*.

Grathwohl, W., Choi, D., Wu, Y., Roeder, G., and Duvenaud, D. (2017). Backpropagation through the void: Optimizing control variates for black-box gradient estimation. *Arxiv preprint arXiv:1711.00123*.

Graves, A. (2011). Practical variational inference for neural networks. In Shawe-Taylor, J., Zemel, R. S., Bartlett, P. L., Pereira, F. C. N., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 24: 25th Annual Conference on Neural Information Processing Systems 2011. Proceedings of a meeting held 12-14 December 2011, Granada, Spain.*, pages 2348–2356.

Graves, A. (2013). Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*.

Graves, A., Mohamed, A.-r., and Hinton, G. (2013). Speech recognition with deep recurrent neural networks. In *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on*, pages 6645–6649. IEEE.

Greensmith, E., Bartlett, P. L., and Baxter, J. (2004). Variance reduction techniques for gradient estimates in reinforcement learning. *The Journal of Machine Learning Research*, 5:1471–1530.

Gregor, K., Besse, F., Rezende, D. J., Danihelka, I., and Wierstra, D. (2016). Towards conceptual compression. In Lee et al. (2016), pages 3549–3557.

Gutmann, M. and Hyvärinen, A. (2010). Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In *International Conference on Artificial Intelligence and Statistics*, pages 297–304.

Guyon, I., von Luxburg, U., Bengio, S., Wallach, H. M., Fergus, R., Vishwanathan, S. V. N., and Garnett, R., editors (2017). *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*.

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

Hardt, M., Recht, B., and Singer, Y. (2016). Train faster, generalize better: Stability of stochastic gradient descent. In Balcan and Weinberger (2016), pages 1225–1234.

He, K., Gkioxari, G., Dollár, P., and Girshick, R. (2017). Mask r-cnn. *arXiv preprint arXiv:1703.06870*.

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

He, K., Zhang, X., Ren, S., and Sun, J. (2015b). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Identity mappings in deep residual networks. *CoRR*, abs/1603.05027.

Heskes, T. M. and Kappen, B. (1993). On-line learning processes in artificial neural networks. *North-Holland Mathematical Library*, 51:199–233.

Hinton, G., Deng, L., Yu, D., Dahl, G. E., Mohamed, A.-r., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Sainath, T. N., et al. (2012). Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97.

Hinton, G. E. (1984). Distributed representations.

Hinton, G. E. and Van Camp, D. (1993). Keeping the neural networks simple by minimizing the description length of the weights. In *Proceedings of the sixth annual conference on Computational learning theory*, pages 5–13. ACM.

Hinton, G. E. and Zemel, R. S. (1993). Autoencoders, minimum description length and helmholtz free energy. In Cowan, J. D., Tesauro, G., and Alspector, J., editors, *Advances in Neural Information Processing Systems 6, [7th NIPS Conference, Denver, Colorado, USA, 1993]*, pages 3–10. Morgan Kaufmann.

Hochreiter, S. (1991). Untersuchungen zu dynamischen neuronalen netzen. *Diploma, Technische Universität München*, 91.

Hochreiter, S. and Schmidhuber, J. (1997a). Flat minima. *Neural Computation*, 9(1):1–42.

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

Hoffman, M. D., Blei, D. M., Wang, C., and Paisley, J. (2013). Stochastic variational inference. *The Journal of Machine Learning Research*, 14(1):1303–1347.

Honkela, A. and Valpola, H. (2004). Unsupervised variational bayesian learning of nonlinear models. In *Advances in neural information processing systems*, pages 593–600.

Hornik, K. (1991). Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257.

Hyvärinen, A., Karhunen, J., and Oja, E. (2004). *Independent component analysis*, volume 46. John Wiley & Sons.

Hyvärinen, A. and Pajunen, P. (1999). Nonlinear independent component analysis: Existence and uniqueness results. *Neural Networks*, 12(3):429–439.

Im, D. J., Tao, M., and Branson, K. (2016). An empirical analysis of deep network loss surfaces. *arXiv preprint arXiv:1612.04010*.

Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Bach and Blei (2015), pages 448–456.

Jacobsen, J.-H., Smeulders, A., and Oyallon, É. (2018). i-revnet: Deep invertible networks. *International Conference on Learning Representations*.

Jarrett, K., Kavukcuoglu, K., LeCun, Y., et al. (2009). What is the best multi-stage architecture for object recognition? In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 2146–2153. IEEE.

Kavukcuoglu, K., Sermanet, P., Boureau, Y.-L., Gregor, K., Mathieu, M., and Cun, Y. L. (2010). Learning convolutional feature hierarchies for visual recognition. In *Advances in neural information processing systems*, pages 1090–1098.

Keskar, N. S., Mudigere, D., Nocedal, J., Smelyanskiy, M., and Tang, P. T. P. (2017). On large-batch training for deep learning: Generalization gap and sharp minima. In *ICLR'2017, arXiv:1609.04836*.

Kingma, D. P. and Ba, J. (2015). Adam: A method for stochastic optimization. In *ICLR'2015, arXiv:1412.6980*.

Kingma, D. P., Salimans, T., Józefowicz, R., Chen, X., Sutskever, I., and Welling, M. (2016). Improving variational autoencoders with inverse autoregressive flow. In Lee et al. (2016), pages 4736–4744.

Kingma, D. P. and Welling, M. (2014a). Auto-encoding variational bayes. In *ICLR'2014, arXiv:1312.6114*.

Kingma, D. P. and Welling, M. (2014b). Efficient gradient-based inference through transformations between bayes nets and neural nets. In Xing and Jebara (2014), pages 1782–1790.

Kleijnen, J. P. and Rubinstein, R. Y. (1996). Optimization and sensitivity analysis of computer simulation models by the score function method. *European Journal of Operational Research*, 88(3):413–427.

Klyachko, A. A. (2000). Random walks on symmetric spaces and inequalities for matrix spectra. *Linear Algebra and its Applications*, 319(1-3):37–59.

Knowles, S. (2017). A letter to my teenage self. *Teen Vogue*.

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

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105.

Krueger, D., Ballas, N., Jastrzebski, S., Arpit, D., Kanwal, M. S., Maharaj, T., Bengio, E., Fischer, A., and Courville, A. (2017). Deep nets don't learn via memorization.

Kullback, S. and Leibler, R. A. (1951). On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86.

Lafond, J., Vasilache, N., and Bottou, L. (2016). About diagonal rescaling applied to neural nets. *ICML Workshop on Optimization Methods for the Next Generation of Machine Learning*.

Langford, J. and Caruana, R. (2001). (not) bounding the true error. In Dietterich, T. G., Becker, S., and Ghahramani, Z., editors, *Advances in Neural Information Processing Systems 14 [Neural Information Processing Systems: Natural and Synthetic, NIPS 2001, December 3-8, 2001, Vancouver, British Columbia, Canada]*, pages 809–816. MIT Press.

Langford, J. and Pineau, J., editors (2012). *Proceedings of the 29th International Conference on Machine Learning, ICML 2012, Edinburgh, Scotland, UK, June 26 - July 1, 2012*. icml.cc / Omnipress.

Larochelle, H. and Murray, I. (2011). The neural autoregressive distribution estimator. In *International Conference on Artificial Intelligence and Statistics*, pages 29–37.

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

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

LeCun, Y. and Bengio, Y. (1994). Word-level training of a handwritten word recognizer based on convolutional neural networks. In *Pattern Recognition, 1994. Vol. 2-Conference B: Computer Vision & Image Processing., Proceedings of the 12th IAPR International. Conference on*, volume 2, pages 88–92. IEEE.

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

LeCun, Y., Cortes, C., and Burges, C. J. (1998b). The mnist database of handwritten digits.

Lee, C.-Y., Xie, S., Gallagher, P., Zhang, Z., and Tu, Z. (2014). Deeply-supervised nets. *arXiv preprint arXiv:1409.5185*.

Lee, D. D., Sugiyama, M., von Luxburg, U., Guyon, I., and Garnett, R., editors (2016). *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*.

Levy, D., Hoffman, M. D., and Sohl-Dickstein, J. (2017). Generalizing hamiltonian monte carlo with neural networks. *arXiv preprint arXiv:1711.09268*.

Li, H., Xu, Z., Taylor, G., and Goldstein, T. (2017). Visualizing the loss landscape of neural nets. *arXiv preprint arXiv:1712.09913*.

Liang, T., Poggio, T., Rakhlin, A., and Stokes, J. (2017). Fisher-rao metric, geometry, and complexity of neural networks. *CoRR*, abs/1711.01530.

Liang-Chieh, C., Papandreou, G., Kokkinos, I., murphy, k., and Yuille, A. (2015). Semantic image segmentation with deep convolutional nets and fully connected crfs. In *ICLR'2015, arXiv:1412.7062*.

Liese, F. and Vajda, I. (2006). On divergences and informations in statistics and information theory. *IEEE Transactions on Information Theory*, 52(10):4394–4412.

Lin, T., Horne, B. G., Tino, P., and Giles, C. L. (1996). Learning long-term dependencies in narx recurrent neural networks. *IEEE Transactions on Neural Networks*, 7(6):1329–1338.

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

Louizos, C. and Welling, M. (2017). Multiplicative normalizing flows for variational bayesian neural networks. *arXiv preprint arXiv:1703.01961*.

Mandt, S., Hoffman, M. D., and Blei, D. M. (2016). A variational analysis of stochastic gradient algorithms. In Balcan and Weinberger (2016), pages 354–363.

Martens, J. and Grosse, R. (2015). Optimizing neural networks with kronecker-factored approximate curvature. In *International Conference on Machine Learning*, pages 2408–2417.

Martens, J. and Medabalimi, V. (2014). On the expressive efficiency of sum product networks. *arXiv preprint arXiv:1411.7717*.

McAllester, D. A. (1999). Pac-bayesian model averaging. In *Proceedings of the twelfth annual conference on Computational learning theory*, pages 164–170. ACM.

McLachlan, G. J., Krishnan, T., and Ng, S. K. (2004). The em algorithm. Technical report, Papers/Humboldt-Universität Berlin, Center for Applied Statistics and Economics (CASE).

Menezes, A. J., Van Oorschot, P. C., and Vanstone, S. A. (1996). *Handbook of applied cryptography*. CRC press.

Mitchell, T. M. (1997). Machine learning. 1997. *Burr Ridge, IL: McGraw Hill*, 45.

Mnih, A. and Gregor, K. (2014). Neural variational inference and learning in belief networks. In Xing and Jebara (2014), pages 1791–1799.

Mohamed, S. and Lakshminarayanan, B. (2016). Learning in implicit generative models. *arXiv preprint arXiv:1610.03483*.

Montufar, G. F., Pascanu, R., Cho, K., and Bengio, Y. (2014). On the number of linear regions of deep neural networks. In *Advances in neural information processing systems*, pages 2924–2932.

Murphy, K. P. (2012). *Machine learning: a probabilistic perspective*. MIT press.

Nair, V. and Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 807–814.

Neal, R. M. (2012). *Bayesian learning for neural networks*, volume 118. Springer Science & Business Media.

Neal, R. M. et al. (2011). Mcmc using hamiltonian dynamics. *Handbook of Markov Chain Monte Carlo*, 2(11).

Neal, R. M. and Hinton, G. E. (1998). A view of the em algorithm that justifies incremental, sparse, and other variants. In *Learning in graphical models*, pages 355–368. Springer.

Nesterov, Y. et al. (2007). Gradient methods for minimizing composite objective function.

Nesterov, Y. and Vial, J.-P. (2008). Confidence level solutions for stochastic programming. *Automatica*, 44(6):1559–1568.

Netzer, Y., Wang, T., Coates, A., Bissacco, A., Wu, B., and Ng, A. Y. (2011). Reading digits in natural images with unsupervised feature learning. In *NIPS workshop on deep learning and unsupervised feature learning*, volume 2011, page 5. Granada, Spain.

Neyshabur, B., Bhojanapalli, S., McAllester, D., and Srebro, N. (2017). Exploring generalization in deep learning. In Guyon et al. (2017), pages 5949–5958.

Neyshabur, B., Salakhutdinov, R. R., and Srebro, N. (2015). Path-sgd: Path-normalized optimization in deep neural networks. In *Advances in Neural Information Processing Systems*, pages 2422–2430.

Paisley, J. W., Blei, D. M., and Jordan, M. I. (2012). Variational bayesian inference with stochastic search. In Langford and Pineau (2012).

Papamakarios, G., Murray, I., and Pavlakou, T. (2017). Masked autoregressive flow for density estimation. In *Advances in Neural Information Processing Systems*, pages 2335–2344.

Pascanu, R. and Bengio, Y. (2014). Revisiting natural gradient for deep networks. *ICLR*.

Pham, D. T. and Garat, P. (1997). Blind separation of mixture of independent sources through a quasi-maximum likelihood approach. *IEEE Trans. Signal Processing*, 45(7):1712–1725.

Poole, B., Lahiri, S., Raghu, M., Sohl-Dickstein, J., and Ganguli, S. (2016). Exponential expressivity in deep neural networks through transient chaos. In *Advances In Neural Information Processing Systems*, pages 3360–3368.

Precup, D. and Teh, Y. W., editors (2017). *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*. PMLR.

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

Raghu, M., Poole, B., Kleinberg, J. M., Ganguli, S., and Sohl-Dickstein, J. (2017). On the expressive power of deep neural networks. In Precup and Teh (2017), pages 2847–2854.

Rainforth, T., Le, T. A., Maddison, M. I. C. J., and Wood, Y. W. T. F. (2017). Tighter variational bounds are not necessarily better.

Rezende, D. J. and Mohamed, S. (2015). Variational inference with normalizing flows. In Bach and Blei (2015), pages 1530–1538.

Rezende, D. J., Mohamed, S., and Wierstra, D. (2014). Stochastic backpropagation and approximate inference in deep generative models. In Xing and Jebara (2014), pages 1278–1286.

Rippel, O. and Adams, R. P. (2013). High-dimensional probability estimation with deep density models. *arXiv preprint arXiv:1302.5125*.

Robbins, H. and Monro, S. (1951). A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1988). Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., et al. (2015). Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252.

Sagun, L., Bottou, L., and LeCun, Y. (2016). Singularity of the hessian in deep learning. *arXiv preprint arXiv:1611.07476*.

Sagun, L., Güney, V. U., Arous, G. B., and LeCun, Y. (2014). Explorations on high dimensional landscapes. *arXiv preprint arXiv:1412.6615*.

Salakhutdinov, R. and Hinton, G. E. (2009). Deep boltzmann machines. In *International Conference on Artificial Intelligence and Statistics*, pages 448–455.

Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A., and Chen, X. (2016). Improved techniques for training gans. In *Advances in Neural Information Processing Systems*, pages 2226–2234.

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

Sashank J. Reddi, Satyen Kale, S. K. (2018). On the convergence of adam and beyond. *International Conference on Learning Representations*. accepted as oral presentation.

Saul, L. K., Jaakkola, T., and Jordan, M. I. (1996). Mean field theory for sigmoid belief networks. *Journal of artificial intelligence research*, 4(1):61–76.

Saxe, A. M., McClelland, J. L., and Ganguli, S. (2013). Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *CoRR*, abs/1312.6120.

Simonyan, K. and Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition. In *ICLR'2015, arXiv:1409.1556*.

Smith, S. L. and Le, Q. V. (2017). A bayesian perspective on generalization and stochastic gradient descent.

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

Sohl-Dickstein, J. (2012). The natural gradient by analogy to signal whitening, and recipes and tricks for its use. *arXiv preprint arXiv:1205.1828*.

Solomonoff, R. J. (1978). Complexity-based induction systems: comparisons and convergence theorems. *Information Theory, IEEE Transactions on*, 24(4):422–432.

Song, J., Zhao, S., and Ermon, S. (2017). A-nice-mc: Adversarial training for mcmc. In *Advances in Neural Information Processing Systems*, pages 5146–5156.

Srivastava, R. K., Greff, K., and Schmidhuber, J. (2015). Training very deep networks. In *Advances in neural information processing systems*, pages 2377–2385.

Sussillo, D. and Abbott, L. (2014). Random walk initialization for training very deep feedforward networks. *arXiv preprint arXiv:1412.6558*.

Susskind, J. M., Anderson, A. K., and Hinton, G. E. (2010). The toronto face database. *Department of Computer Science, University of Toronto, Toronto, ON, Canada, Tech. Rep.*

Sutskever, I., Martens, J., Dahl, G., and Hinton, G. (2013). On the importance of initialization and momentum in deep learning. In *Proceedings of the 30th international conference on machine learning (ICML-13)*, pages 1139–1147.

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112.

Sweldens, W. (1998). The lifting scheme: A construction of second generation wavelets. *SIAM journal on mathematical analysis*, 29(2):511–546.

Swirszcz, G., Czarnecki, W. M., and Pascanu, R. (2016). Local minima in training of deep networks. *CoRR*, abs/1611.06310.

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

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

Tabak, E. and Turner, C. V. (2013). A family of nonparametric density estimation algorithms. *Communications on Pure and Applied Mathematics*, 66(2):145–164.

Telgarsky, M. (2016). benefits of depth in neural networks. In Feldman, V., Rakhlin, A., and Shamir, O., editors, *Proceedings of the 29th Conference on Learning Theory, COLT 2016, New York, USA, June 23-26, 2016*, volume 49 of *JMLR Workshop and Conference Proceedings*, pages 1517–1539. JMLR.org.

Theis, L. and Bethge, M. (2015). Generative image modeling using spatial lstms. In *Advances in Neural Information Processing Systems*, pages 1918–1926.

Theis, L., van den Oord, A., and Bethge, M. (2016). A note on the evaluation of generative models. In *ICLR'2016 arXiv:1511.01844*.

Tieleman, T. and Hinton, G. (2012). Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural Networks for Machine Learning*, 4.

Tomczak, J. M. and Welling, M. (2016). Improving variational auto-encoders using householder flow. *arXiv preprint arXiv:1611.09630*.

Tomczak, J. M. and Welling, M. (2017). Improving variational auto-encoders using convex combination linear inverse autoregressive flow. *arXiv preprint arXiv:1706.02326*.

Tucker, G., Mnih, A., Maddison, C. J., Lawson, J., and Sohl-Dickstein, J. (2017). Rebar: Low-variance, unbiased gradient estimates for discrete latent variable models. In *Advances in Neural Information Processing Systems*, pages 2624–2633.

Uria, B., Murray, I., and Larochelle, H. (2013). Rnade: The real-valued neural autoregressive density-estimator. In *Advances in Neural Information Processing Systems*, pages 2175–2183.

Valpola, H. and Karhunen, J. (2002). An unsupervised ensemble learning method for nonlinear dynamic state-space models. *Neural computation*, 14(11):2647–2692.

van den Oord, A., Kalchbrenner, N., Espeholt, L., Kavukcuoglu, K., Vinyals, O., and Graves, A. (2016a). Conditional image generation with pixelcnn decoders. In Lee et al. (2016), pages 4790–4798.

van den Oord, A., Kalchbrenner, N., and Kavukcuoglu, K. (2016b). Pixel recurrent neural networks. In Balcan and Weinberger (2016), pages 1747–1756.

van den Oord, A., Li, Y., Babuschkin, I., Simonyan, K., Vinyals, O., Kavukcuoglu, K., Driessche, G. v. d., Lockhart, E., Cobo, L. C., Stimberg, F., et al. (2017). Parallel wavenet: Fast high-fidelity speech synthesis. *arXiv preprint arXiv:1711.10433*.

van den Oord, A. and Schrauwen, B. (2014). Factoring variations in natural images with deep gaussian mixture models. In *Advances in Neural Information Processing Systems*, pages 3518–3526.

Vapnik, V. (1992). Principles of risk minimization for learning theory. In *Advances in neural information processing systems*, pages 831–838.

Vapnik, V. (2013). *The nature of statistical learning theory*. Springer science & business media.

Vapnik, V. N. and Chervonenkis, A. Y. (2015). On the uniform convergence of relative frequencies of events to their probabilities. In *Measures of complexity*, pages 11–30. Springer.

Wainwright, M. J., Jordan, M. I., et al. (2008). Graphical models, exponential families, and variational inference. *Foundations and Trends® in Machine Learning*, 1(1–2):1–305.

Welling, M. and Teh, Y. W. (2011). Bayesian learning via stochastic gradient langevin dynamics. In Getoor, L. and Scheffer, T., editors, *Proceedings of the 28th International Conference on Machine Learning, ICML 2011, Bellevue, Washington, USA, June 28 - July 2, 2011*, pages 681–688. Omnipress.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256.

Wilson, A. C., Roelofs, R., Stern, M., Srebro, N., and Recht, B. (2017). The marginal value of adaptive gradient methods in machine learning. In Guyon et al. (2017), pages 4151–4161.

Wilson, D. R. and Martinez, T. R. (2003). The general inefficiency of batch training for gradient descent learning. *Neural Networks*, 16(10):1429–1451.

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

Wu, Y., Burda, Y., Salakhutdinov, R., and Grosse, R. (2017). On the quantitative analysis of decoder-based generative models. In *ICLR'2017, arXiv:1611.04273*.

Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., et al. (2016). Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*.

Xing, E. P. and Jebara, T., editors (2014). *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014*, volume 32 of *JMLR Workshop and Conference Proceedings*. JMLR.org.

Yu, F. and Koltun, V. (2016). Multi-scale context aggregation by dilated convolutions. In *ICLR'2016, arXiv:1511.07122*.

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

Zaremba, W., Sutskever, I., and Vinyals, O. (2014). Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*.

Zhang, C., Bengio, S., Hardt, M., Recht, B., and Vinyals, O. (2017). Understanding deep learning requires rethinking generalization. In *ICLR'2017, arXiv:1611.03530*.

Zhou, Y. and Chellappa, R. (1988). Computation of optical flow using a neural network. In *IEEE International Conference on Neural Networks*, volume 1998, pages 71–78.