Thesis for the first year of the Applied Mathematics Master of University Paris Dauphine

Title : Adversarial Contrastive Estimation

By : HADNI Yazid, ALAKA Christian, EYROLLE Nicolas

Supervisor : Christian Robert
Group number : 5

Confidentiality : ☑ No ☐ Yes (Length: ☐ 1 year ☐ 2 years)

Université Paris-Dauphine – MIDO – Bureau B536, Place du Maréchal de Lattre de Tassigny, 75016 PARIS

Contents

Table of contents **3**

1 Introduction **5**

2 Noise Contrastive Estimation **7**
2.1 Framework 7
2.2 From estimation to classification: from unsupervised to supervised learning 8
2.3 Choice of hyperparameters 9
2.4 Numerical Experiments 10
2.4.1 The impact of $\nu = \frac{dataset's\ size}{noisesample'ssize}$: 10
2.4.2 Criterion for closeness between noise and data 12

3 GAN **15**
3.1 Theoretical background 15
3.2 Link between GAN and NCE 17
3.3 Parametric GAN 18
3.3.1 One dimensional Case 18
3.3.2 Multidimensional Case 19
3.4 Non Parametric GAN 19
3.4.1 Neural Net theoretical background 19
3.4.2 Kernel Density estimation theoretical background 21
3.4.3 Non parametric GAN Estimation 23
3.5 GAN vs NCE 28

4 Conclusion **29**

A Appendix NCE : Figures 6, 7 and 8 **31**

Chapter 1

Introduction

A statistical model with probability density function $p(x;\theta)$ is said to be unnormalized if its partition function $Z(\theta) = \int p(x;\theta)d\mu(x)$ (i.e the normalizing constant) does not equal one. In general it cannot be calculated explicitly, or is difficult to compute in practice and is therefore unknown.

This kind of models are encountered in many applications, for example Markov networks [4], Boltzmann machines [12], neural probabilistic language models [17]. Thus, looking for estimation methods for unnormalized models is an important research topic in many fields .

Following Gutmann and Hyvärinen's categorisation [10] we can identify two main categories of estimation methods for unnormalized models: methods which approximate the partition function and methods which do not because of its high computation cost. This segmentation corresponds to a trade-off between statistical and computational efficiency. Indeed, approximating the model's partition function $Z(\theta)$ allows to convert an unnormalized model into a normalized one by dividing its normalizing constant with respect to the following relation : $\int \frac{p(x;\theta)}{Z(\theta)}d\mu(x) = 1$. It is useful as we thus can apply basic estimation principles for normalized model such as Maximum Likelihood Estimation. However, $Z(\theta)$'s approximation relies on numerical computation based on sampling methods such as Monte Carlo . Thus, as soon as the dimension increases, these methods require a huge amount of simulations which leads to expensive numerical computations. This phenomenon is known as the "curse of dimensionality". That's why several methods avoiding the normalizing constant computations have been suggested.

Among them, two important methods seem to distinguish from the others because of their statistical and computational efficiency : Score Matching and Noice Contrastive Estimation respectively published by Hyvärinen [13] and Gutmann and Hyvärinen [10].

Score Matching (SM) defines a distance over parameters based on score function. The good estimate is the one which minimizes this distance. The score function has the advantage of getting rid of the normalizing constant and then overcome the partition function computations' drawbacks the above mentioned methods are exposed to.

However this method presents two main inconvenients: it requires the pdf to be smooth enough (i.e does not apply in all cases) and does not provide any constant's estimation as it gets rid of it. This can be problematic as estimating normalizing constant's value can be useful like in Bayesian framework to deal with model selection where evidence's computation is necessary (see [19] for more details).

Meanwhile, NCE relies on transforming the estimation problem to a classification between the dataset and some generated noise sample. It faces normalizing constant's approximation by including it among model's parameters. Since it does not require the pdf to be smooth, even if it relies on sampling noise (and so could be less computational efficient compared to SM), it overcomes in a certain way the SM's aforementioned drawbacks.

As we are going to see further in this paper, according to Gutmann and Hyvärinen's intuition, noise's choice has an impact on estimation's precision. The more the confusion reigns between data and noise samples, the better the estimation is. This leads us to propose to augment the confusion in NCE with an adversarial learned adaptive sampler that finds harder examples to distinguish from dataset. We then take inspiration from the Generative Adversarial Network (GAN) [7] framework to propose a new contrastive estimation scheme. Similar work has recently been done in the case of language modelling , in order to learn a discrete distribution over a vocabulary of words[5], and it has been called Adversarial Contrastive Estimation (ACE). To the best of our knowledge , this idea has not yet been applied to continuous distribution as we do in this work for the estimation of the partition function.

This paper is organized as follows:

1. We describe NCE methods and see the advantages it has for the normalizing constant estimation.
2. We illustrate theoretical results by several normalizing constant estimation experiences.
3. We describe GANs framework and see how to adapt it to our estimation scheme.
4. We compare the performances between NCE and GAN method in order to estimate the normalizing constant.

This work is mainly based on two article we have already mentioned : [10] and [7].

All experiments we performed are available at : `https://github.com/alakachr/NCE_GAN`.

Chapter 2

Noise Contrastive Estimation

2.1 Framework

Let $\mathbf{x} \in \mathbb{R}^n$ be a random vector that follows an unknown distribution with density function $p_d(.)$. This density function is modeled by a family of parameterized functions $\{p_m(.;\theta)\}_\theta$ where θ is a vector of parameters' space Θ. We assume that $p_d(.)$ belongs to this family, which means that there is $\theta^\star$ such that $p_d(.) = p_m(.;\theta^\star)$.

Our problem therefore consists in estimating θ from the sample by maximizing some objective function. Obviously, any solution $\hat{\theta}$ of the estimation problem must meet the following constraint :

$$\int p_m(\mathbf{u};\hat{\theta})\mathrm{d}\mathbf{u} = 1$$

Theoretically, we can always redefine the density $p_d(.)$ in the following way :

$$p_m(.;\theta) = \frac{p_m^0(.,\theta)}{Z(\theta)} \qquad where \qquad Z(\theta) = \int p_m^0(\mathbf{u};\theta)\mathrm{d}\mathbf{u},$$

Thus $p_m^0(.;\theta)$ is the unnormalized part of $p_m(.;\theta)$. As we discussed earlier the computation of $Z(\theta)$, which happens to be the normalizing constant can be difficult to achieve. Therefore, Gutmann and Hyvärinen [11] suggest to consider the normalization constant as another parameter of the model. Consequently, our parameter space is the following $\bar{\Theta} = \{(\theta, Z(\theta)),\ \theta \in \Theta\}$. The normalizing constant can be estimated by maximizing the same objective function. Nevertheless, this approach cannot be used with the maximum likelihood method since if we make $Z(\theta)$ tend towards 0, the likelihood can become arbitrarily large.

2.2 From estimation to classification: from unsupervised to supervised learning

As described previously, the main idea behind NCE is to estimate the parameters by learning to discriminate between the data $\mathbf{x}$ and some artificially generated noise $\mathbf{y} \in \mathbb{R}^{T_n}$ $p_n(.)$. A relative description of our data $\mathbf{x}$ is given by the ratio $\frac{p_d}{p_n}$. Moreover, if we know the differences between $\mathbf{x}$ and $\mathbf{y}$ (as well as the properties that characterize $\mathbf{y}$), we can deduce the properties of $\mathbf{x}$. Heuristically, that would mean if we have p_n, we can obtain p_d thanks to the ratio of the two densities. It is learning by comparison. Indeed, the algorithm is supposed to compare the properties of $\mathbf{x}$ and $\mathbf{y}$ which will give a description of $\mathbf{x}$.

Now let's define NCE more formally and link it to a supervised learning method. Let $U = (u_1, \ldots, u_{T_n+T_d})$ be the union of X and Y and let $C = (C_1, \ldots, C_{T_n+T_d})$ a random vector such that $C_i \overset{i.i.d.}{\sim} Bern(\frac{T_d}{T_n+T_d}), \forall i \in \{1, \ldots, T_n + T_d\}$. In other words,

$$C_i = \begin{cases} 1 & \text{if } u_i \in X \\ 0 & \text{if } u_i \in Y \end{cases}$$

Since the parameters of our density function $p_d(.)$ are unknown, the conditional probability $p(. \mid C = 1)$ is modeled by $p_m(.;\theta)$, where, $p_m(.;\theta)$ would be a normalized density function (with the normalizing factor included in the parameter space). The conditional probability densities are therefore

$$p(u \mid C = 1; \theta) = p_m(u; \theta); \;\; p(u \mid C = 0) = p_n(u). \tag{2.1}$$

According to the Bayes rules we get :

$$P(C = 1 \mid u; \theta) = \frac{p_m(u;\theta)}{p_m(u;\theta) + \nu p_n(u)} := h(u;\theta) \quad where \; \nu = \frac{T_n}{T_d} \tag{2.2}$$

$$P(C = 0 \mid u; \theta) = 1 - h(u;\theta) \tag{2.3}$$

With a little computation we get:

$$h(u;\theta) = \frac{1}{1 + \nu \exp(-G(u;\theta))}, \tag{2.4}$$

$$G(u;\theta) = \ln(p_m(u;\theta)) - \ln(p_n(u)) \tag{2.5}$$

We recognize the sigmoid function associated to logistic regression.

Right now, the model we are dealing with, is a Bernoulli model with parameters $p(\theta, Z(\theta)) = h(u;\theta)$, depending on the parameters of our unnormalized initial model. Thus the model likelihood is the following:

$$\mathcal{L}(\theta; u) = \Pi_{i=1}^{T_n+T_d} p^{C_i}(1-p)^{1-C_i} \tag{2.6}$$

Thus we obtain the log-likelihood :

$$\ell(\theta;u) = \sum_{i=1}^{T_n+T_d} C_i \ln(h(u_i;\theta)) + (1-C_i)\ln(1-h(u_i;\theta)) = \sum_{i=1}^{T_d} \ln(h(x_i;\theta)) + \sum_{i=1}^{T_n} \ln(1-h(y_i;\theta)) \tag{2.7}$$

In order to get a good estimation of θ we can apply the MLE principle to our new normalized model. Let's give the formal definition of the NCE estimator:

$$\theta_{NCE} := \underset{\theta}{argmax} \frac{1}{T_d}\ell(\theta;u) = \frac{1}{T_d}\sum_{i=1}^{T_d} \ln(h(x_i;\theta)) + \frac{\nu}{T_n}\sum_{i=1}^{T_n} \ln(1-h(y_i;\theta)) \tag{2.8}$$

We remark that, the only factors (hyperparameters) we can play on in order to influence estimation are :

1. Noise distribution
2. Noise sample size

2.3 Choice of hyperparameters

In order to get an estimator with a low variance, Gutmann and Hyvarynen in [10] advocate to choose a noise which respects the following properties:

1. Choose noise for which an analytical expression for $ln(p_n)$ is available. This condition is necessary to implement logistic regression.
2. Choose noise that can be sampled easily. This allows to make the sample size as large as possible.
3. Make the noise sample size as large as computationally possible.
4. Choose noise that is in some aspect, for example with respect to its covariance structure, similar to the data.

The motivation for the first two is trivial. We are going to present the theoretical result that justify the last two [10].

Theorem 2.3.1 (Asymptotic normality)**.** *$\sqrt{T_d}(\hat{\theta} - \theta^\star)$ is asymptotically normal with mean zero and covariance matrix Σ such that:*

$$\Sigma = \mathcal{I}_\nu - (1+\frac{1}{\nu})\mathcal{I}_\nu^{-1} E\left(P_\nu g\right) E\left(P_\nu g\right)^T \mathcal{I}_\nu^{-1} \tag{2.9}$$

where $g(u) = \nabla_\theta \ln p_m(u,,\theta)|_{\theta^\star}$, $P_\nu = \frac{\nu p_n(u)}{p_d(u)+\nu p_n(u)}$, $\mathcal{I}_\nu = \int g(u)g(u)^T P_\nu(u) p_d(u) du$ and $E(P_\nu g) = \int P_\nu(u) g(u) p_d(u) du$

Corollary 2.3.1. *For $\nu \to \infty$ Σ is independent of the choice of p_n and equals*

$$\Sigma = \mathcal{I}^{-1} - \mathcal{I}^{-1} E\left(g\right) E\left(g\right)^T \mathcal{I}^{-1} \tag{2.10}$$

Proof. When $\nu \to +\infty$, $P_\nu = \frac{\nu p_n(u)}{p_d(u)+\nu p_n(u)} \to 1$. So nothing depends on ν. We then get the above limit.

Remark. *For normalized models, it follows that NCE's estimator is Fisher efficient when $\nu \to \infty$. Indeed, in Corollary 2.3.1 g equals the score function and then $E(Pg) = 0$. Finally, $\mathcal{I}$ corresponds to the Fisher information's matrix. Thus the estimator's variance is equal to the inverse of the information matrix which is the Cramer-Rao's bound.*

Corollary 2.3.2. *If $p_n = p_d$ then*

$$\Sigma = (1 + \frac{1}{\nu})(\mathcal{I}^{-1} - \mathcal{I}^{-1} E\left(g\right) E\left(g\right)^T \mathcal{I}^{-1}) \tag{2.11}$$

Proof. When $p_d = p_n$ we have $P_\nu = \frac{\nu}{1+\nu}$ a direct factorization gives the result.

The above corollaries give us the asymptotic behavior of NCE estimator's variance depending on:

1. The choice of the noise sample size distribution (corollary 2.3.1)
2. The choice of the noise distribution (corollary 2.3.2)

Corollary 2.3.1 shows that for a large value of ν the estimator's variance does not depend on the noise distribution. However, it doesn't tell us if the convergence in ν is monotone. Indeed, we have no idea on the relationship which exist between the variance and ν for $\nu < +\infty$ (cf. (2.9)). Also, we have no theoretical guarantees on rate of convergence. The same problem applies to corollary 2.3.2 : we are told to choose noise which is close to the data, but with no theoretical guarantee about a closeness criterion. We don't know how much close to the data is enough to reduce variance. And we don't even know if the optimal noise distribution is $p_n = p_{data}$: minimizing the MSE in the function space with respect to p_n is a difficult problem.

2.4 Numerical Experiments

2.4.1 The impact of $\nu = \frac{dataset's\ size}{noisesample's size}$:

We study the impact of ν on the log-MSE for the estimation of a standard Gaussian for different choice of noise :

1. noise distribution = data distribution
2. noise distribution close to data
3. noise distribution very different from data

4. noise distribution from a different distribution family

The normalizing constant is given by $Z(\mu_{data} = 0, \sigma_{data} = 1) = \frac{1}{\sqrt{2\pi\sigma^2_{data}}}$.

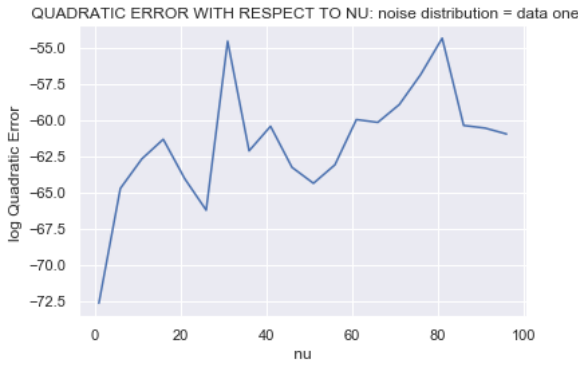


Figure 2.1 – Nu impact on MSE when noise distribution and data one are equal

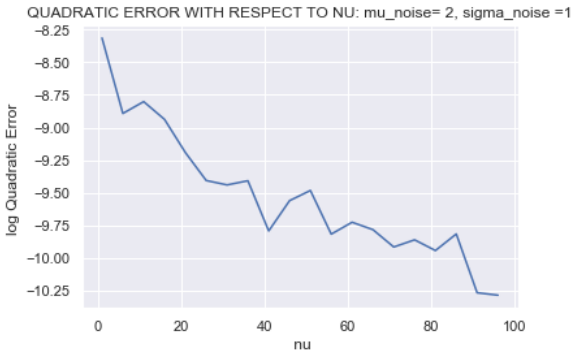


Figure 2.2 – Nu impact on MSE when noise distribution and data one differ slightly

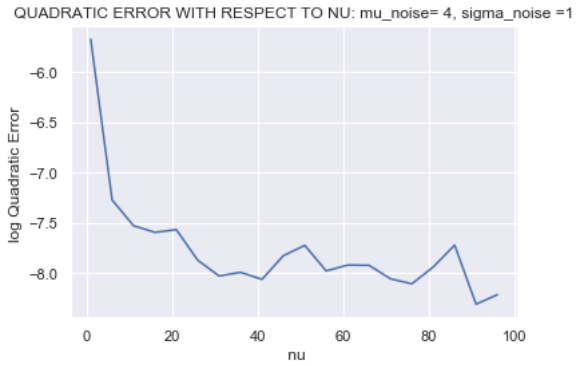


Figure 2.3 – Nu impact on MSE when noise distribution and data one differ a lot

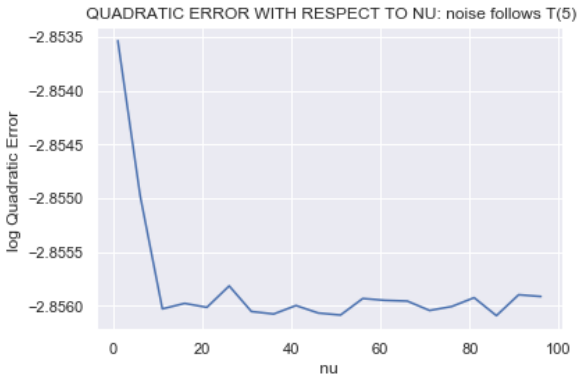


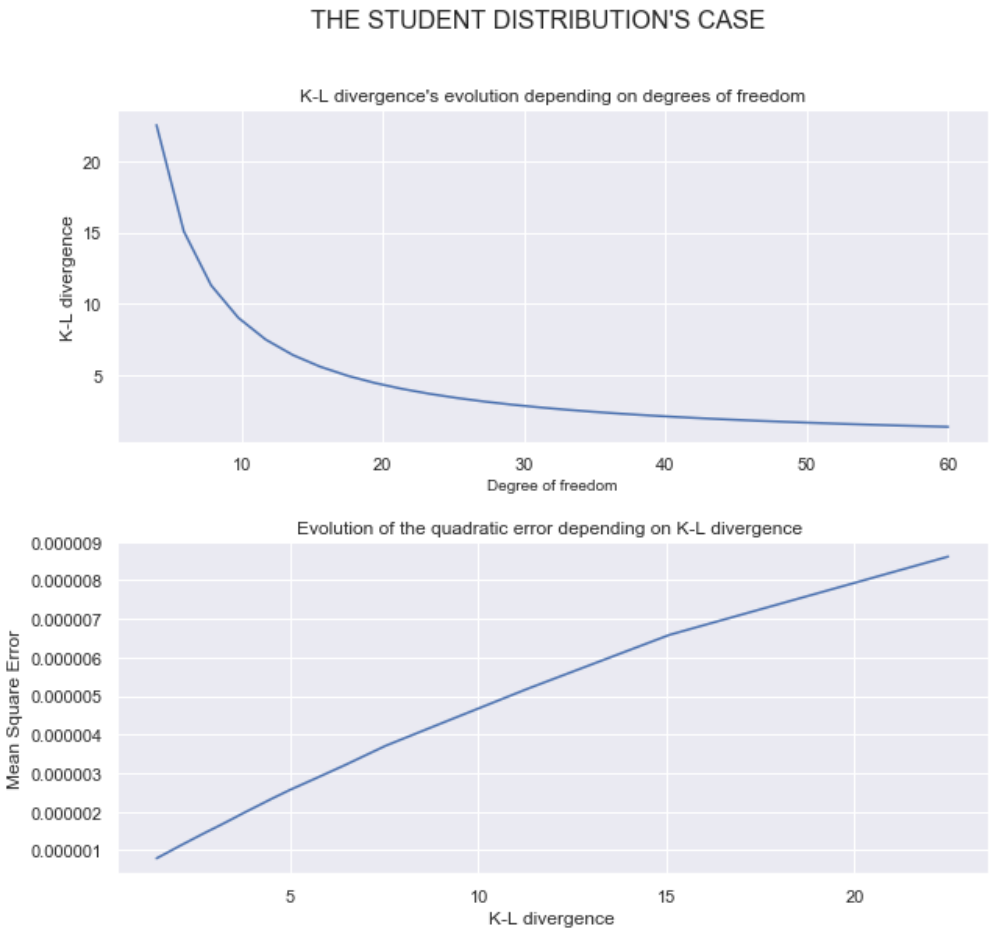
Figure 2.4 – Nu impact on MSE when noise distribution are not in same distribution family. Noise $\sim$ T(20) (low K-L divergence)

In Figures 2.2, 2.3 and 2.4 experiments show that for a noise different from the data, the mean square error decreases with respect to ν. However even if it is not the case when $p_{noise} = p_{data}$ we remark the log MSE is much lower than in other cases(see Figure 2.1).
This confirms the above discussion. Indeed, when noise's sample size is arbitrarily large compared to the data, we get a large amount of noise realization in the area where the data density is large. It then increases the confusion between both samples.
These observations coincide with the one done in [11] for a multivariate gaussian model.

2.4.2 Criterion for closeness between noise and data

Again we choose a one dimensional Gaussian for the data in this experiment. Our goal is to study the impact of a distance measure between distribution like the KL divergence on NCE estimation. We first consider a Student distribution for the noise. An interesting property of the Student law is the fact that it converges (in distribution) to a Gaussian, and this convergence depends only on the degree of freedom. Therefore when the degree of freedom increase, the KL divergence between noise and data decrease.

Figure 2.5 – Kullback Leibler divergence in with noise following a student distribution



The above graphs in figure 2.5 shows that the MSE strictly increase with respect to the Kullback-Leibler divergence. This seems to indicate that a distance measure between distribution could characterize the distance between noise and data with respect to quality of NCE estimation .

In figure A.1, we consider different Gaussian $N(\mu, \sigma)$ for the noise, We first explore how the MSE varies for different choices of μ with σ fixed. We observe that when σ is low, the behaviour of KL divergence and μ with respect to the MSE coincide. However, when σ becomes too large, the choice of μ has no impact and therefore the KL divergence is no longer relevant for the MSE.

In figure A.2, we explore how the MSE varies for different choice of σ with μ fixed. Even if, K-L divergence and Mean Square Error evolve in a similar way, inflection points of both graphs do not precisely correspond when noise is centered far from the data. Hence, basing the choice of the noise on distance between distribution could be misleading. We even observe a small contradiction with Guttman and al. recommendation to choose noise similar to the data with respect to its covariance structure. Indeed, if μ is large, the best choice of noise is not the one with the same variance as the data (this means noise variance needs to be large enough to produce observations where the data likelihood is high).

In Figure A.1, we showed that for a given σ_{noise}, the quadratic error reaches its minimum at $\mu_{noise} = \mu_{data} = 0$. It relies on the fact that, for a given σ, gaussian' realizations are concentrated around the mean. Thus, if μ_{data} and μ_{noise} are far from each other, noise and data realizations have different hotspots. It follows that samples are less confused and so easier to classify. This phenomenon is illustrated in figure A.3.

Figures A.1, A.2, A.3 can be found in the appendix.

Chapter 3

GAN

3.1 Theoretical background

Generative Adversarial Network, in short GAN [9], is an unsupervised learning framework which aims to build a generative model capable of generating synthetic data indistinguishable from natural observations. These methods have notably been applied for synthetic image generation, and are arguably the most popular neural generative framework currently. Other deep generative models examples are Restricted Boltzmann Machines (RBM), Variational Auto Encoders [14], or deep belief networks (DBN) .

In order to learn the generative distribution p_g over the data x, the GAN's algorithm trains a generator G and a discriminator D to play a minimax game with two players. The goal of the generator G is to "fool" the discriminator D by producing indistinguishable sample, while D aims to predict correctly if a realization comes from the real data distribution or the synthetic data distribution. To do this, D is defined as being a differentiable function (usually a neural network) which receives an observation as input and returns the probability that it came from the real dataset. G is also a differentiable function (usually a neural network as well), but it receives a latent noise z as input and returns synthetic observations. It is interesting to see G as some sort of inverse cdf, F^{-1} that receives uniform samples as input and generates realization of a random variable with law F. We therefore want for D, to maximize the log probability of the classification problem, and simultaneously for G, to minimize the probability that D is mistaken on synthetic realizations. This consists in solving the following optimization problem:

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}\,(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))] \tag{3.1}$$

To solve this problem, Goodfellow et al. [9] propose the following stochastic gradient descent algorithm , which consist in a gradient ascent and a gradient descent running in parallel :

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets

1: **for** nb training iterations **do**
2: **for** k steps **do**
2: Sample minibatch of m noise samples $\{z_{(1)}, ..., z_{(m)}\}$ from p_z;
2: Sample minibatch of m examples $\{x_{(1)}, ..., x_{(m)}\}$ from p_z;
2: Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(x^{(i)}\right) + \log\left(1 - D\left(G\left(z^{(i)}\right)\right)\right)\right]$$

3: **end for**
3: Sample minibatch of m noise samples $\{z_{(1)}, ..., z_{(m)}\}$ from p_z
3: Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(z^{(i)}\right)\right)\right)$$

4: **end for**

GANs offers some theoretical guarantees. The following theoretical result from [9] tells us that the optimal discriminator for GAN is the same as NCE:

Proposition 3.1.1. *For G fixed, the optimal discriminator D is*

$$D_G^*(\boldsymbol{x}) = \frac{p_{data}(\boldsymbol{x})}{p_{data}(\boldsymbol{x}) + p_g(\boldsymbol{x})} \tag{3.2}$$

Proof. (taken from [1])

$$\begin{aligned}
\arg\max_D V(D,G) &= \arg\max_D E_{u\sim p_t}[\ln D(u)] + E_{u\sim p_n}[\ln(1 - D(u))] \\
&= \arg\max_D \int_u p_d(u) \ln D(u) du + \int_u p_n(u) \ln(1 - D(u)) du \\
&= \arg\max_D \int_u p_d(u) \ln D(u) + p_n(u) \ln(1 - D(u)) du \\
&= \arg\max_D p_d(u) \ln D(u) + p_n(u) \ln(1 - D(u))
\end{aligned}$$

Given u, find the optimal discriminator:

$$\frac{d\left(p_d(u) \ln D(u) + p_n(u) \ln(1 - D(u))\right)}{dD(u)} = p_d(u) \frac{1}{D(u)} - p_n(u) \frac{1}{1 - D(u)}$$

the solution is expressed as the form:

$$D(u) = \frac{p_d(u)}{p_d(u) + p_n(u)} = \frac{1}{1 + \frac{p_n(u)}{p_d(u)}} = \frac{1}{1 + \exp\left(-\ln \frac{p_d(u)}{p_n(u)}\right)}$$

From there we can simplify the objective function to a minimization problem of the criterion

$$C(G) = \max_D V(G,D) \quad = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}}\left[\log \frac{p_{\text{data}}\,(\boldsymbol{x})}{P_{\text{data}}\,(\boldsymbol{x}) + p_g(\boldsymbol{x})}\right] + \mathbb{E}_{\boldsymbol{x}\sim p_g}\left[\log \frac{p_g(\boldsymbol{x})}{p_{\text{data}}\,(\boldsymbol{x}) + p_g(\boldsymbol{x})}\right]$$

,assuming the optimal discriminator have been reached:

Theorem 3.1.1. *The global minimum of the virtual training criterion $C(G)$ is achieved if and only if $p_g = p_{data}$. Minimizing $C(G)$ is equivalent to minimizing the Jensen– Shannon divergence between the model's distribution and the data generating process which is given by*

$$KL\left(p_{data}\,\|\frac{p_{data}\,+p_g}{2}\right) + KL\left(p_g\|\frac{p_{data}\,+p_g}{2}\right) \tag{3.3}$$

Remark. *Minimizing $C(G)$ is equivalent to minimizing $\mathbb{E}_{\boldsymbol{x}\sim p_g}\left[\log \frac{p_g(\boldsymbol{x})}{p_{data}\,(\boldsymbol{x})+p_g(\boldsymbol{x})}\right]$*

Indeed it is equivalent to minimizing the Kullback-Leibler divergence $KL\left(p_g\|\frac{p_{\text{data}}\,+p_g}{2}\right)$ which is always positive and equals 0 if and only if $p_g = p_{data}$

Finally, this proposition gives some guarantees on the convergence of Algorithm 1.

Proposition 3.1.2. *If G and D have enough capacity, and at each step of Algorithm 1, the discriminator is allowed to reach its optimum given G, and p_g is updated so as to improve the criterion*

$$\mathbb{E}_{x\sim p_\omega}\big[\log D_G^*(\boldsymbol{x})\big] + \mathbb{E}_{\boldsymbol{x}\sim p_g}\big[\log\big(1 - D_G^*(\boldsymbol{x})\big)\big]$$

then p_g converges to p_{data}

3.2 Link between GAN and NCE

Recall NCE objective function :

$$\max_{\theta} J(\boldsymbol{\theta}) = \mathrm{E}\{\ln[h(\mathbf{x};\boldsymbol{\theta})]\} + \mathrm{E}\{\ln[1 - h(\mathbf{y};\boldsymbol{\theta})]\} \tag{3.4}$$

It is similar to the GAN objective function

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}\,(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))] \tag{3.5}$$

Actually, if we take

$$D(u) = h(u) = \frac{p_m(\mathbf{u})}{p_m(\mathbf{u}) + p_g(\mathbf{u})}$$

and

$$y = G(z)$$

where p_g : is G's output density, we can see both objective function are the same.

This means NCE can be interpreted as form of GAN with a fixed generator. Goodfellow et al. [7], made this connection and highlighted the inconvenience of using a fixed noise distribution. .We therefore aim to define a new contrastive estimator with a dynamic generator as the one solution of the problem :

$$\max_{cte} \min_{G} \mathbb{E}_{\boldsymbol{x}\sim p_{\mathrm{m}}} \left[\log \frac{cte p_{\mathrm{m}}(\boldsymbol{x})}{cte p_{\mathrm{m}}(\boldsymbol{x}) + p_g(\boldsymbol{x})}\right] + \mathbb{E}_{\boldsymbol{y}\sim p_g} \left[\log \frac{p_g(\boldsymbol{y})}{cte p_{\mathrm{m}}(\boldsymbol{y}) + p_g(\boldsymbol{y})}\right] \tag{3.6}$$

The objective function gradient for the Discriminator (with respect to the normalizing factor, cte) is then

$$\nabla_D = \frac{1}{n}\sum_x \frac{1}{cte} - \frac{p_m(x)}{cte p_m(x) + p_g(x)} - \frac{1}{n}\sum_y \frac{p_m(y)}{cte p_m(y) + p_g(y)} \tag{3.7}$$

Notice that instead of using a multilayered neural network as a discriminator we used $\frac{p_m}{p_m+p_g}$. It is the logistic classifier used in NCE, and it corresponds for a fixed G, to the optimal GAN disriminator.

We then proceed to define the Generator. We will first consider a simpler case where the generator is restricted to a family of distribution . We will refer to this as a parametric GAN.

3.3 Parametric GAN

3.3.1 One dimensional Case

We consider a normal distribution for the data : $x \sim N(\mu_x, \sigma_x)$. The normalization factor we consider is then given by :$\frac{1}{\sigma_x\sqrt{2\pi}}$

We then define a parametric Gaussian generator as $G(z) = \mu_g + \sigma_g z$ with $z \sim N(0,1)$ and $\theta_g = \{\mu_g; \sigma_g\}$.

We compute the Generator objective function gradient with respect to θ_g. But there is an important subtlety here : even if G's parameter appear in the expression of p_g, we are not differentiating p_g with respect to θ_g. In other words, even if θ_g and θ_{p_g} are equal while differentiating the objective function, θ_g is treated as a variable while we treat θ_{p_g} as a constant This is because we are minimizing with respect to G and not with respect to p_g. We then have

$$\nabla_{\theta_g} \frac{1}{n}\sum_y \log\left(\frac{p_g(y)}{cte p_m(y) + p_g(y)}\right) = \frac{1}{n}\sum_y \frac{\sigma_g p'_g(y)}{p_g(y)} - \frac{cte\sigma_g p'_m(y) + \sigma_g p'_g(y)}{cte p_m(y) + p_g(y)}$$

Where $f'(x) = \partial(f)/\partial(x)$ and $y = \mu_g + \sigma_g$

We train this model according to Algorithm 1, with gradient descent and a fixed manually tuned learning rate, to estimate cte and θ_g. We provide some numerical results for this estimation;

Data	True cte value	$\hat{cte}$ estimate	μ_g	σ_g
$x \sim N(24, 7)$	0.05699	0.05729	22.81	7.3559
$x \sim N(0.5, 0.2)$	1.9947	1.9948	0.505	0.274
$x \sim N(0, 1)$	0.3989	0.3989	1.88e-05	0.9999

3.3.2 Multidimensional Case

We consider a (non degenerate) d-multivariate normal distribution for the data : $X \sim N(\mu_x, \Sigma_x)$. with $\mu_x \in \mathbb{R}^d$ .The normalization factor we consider is then given by :

$$\frac{1}{\sqrt{2\pi^d |\Sigma_x|}}$$

We then define a parametric Gaussian generator as

$$\mathbf{G}(\mathbf{Z}) = \boldsymbol{AZ} + \mu \text{ for } \boldsymbol{Z} \sim \mathcal{N}(0, 1) \tag{3.8}$$

here $\mathbf{\Sigma_g} = \boldsymbol{I_d} \boldsymbol{I_d}^{\mathrm{T}}$

We compute the generator objective function gradient with respect to θ_g as we did for the one dimensional case. This is done with Tensorflow Automatic Differentiation

We train this model according to Algorithm 1 to estimate cte and θ_g. We provide some numerical results for this estimation, with the parameters reached by G at convergence;

dimension	True cte value	$\hat{cte}$ estimate	Σ_{data}	Σ_g
$d = 2$	0.07957	0.07957	$\begin{bmatrix} 25 & 39 \\ 39 & 61 \end{bmatrix}$	$\begin{bmatrix} 25 & 39 \\ 39 & 61 \end{bmatrix}$
$d = 2$	0.20943	0.2092	$\begin{bmatrix} 0.8 & 0.25 \\ 0.8 & 0.25 \end{bmatrix}$	$\begin{bmatrix} 0.8 & 0.25 \\ 0.8 & 0.25 \end{bmatrix}$

3.4 Non Parametric GAN

3.4.1 Neural Net theoretical background

We can define a neural net as a composition of differentiable functions. There exists many kinds of neural nets : MLP (Multilayered Perceptron), CNN (Convolutional Neural Net), RNN (Recurrent Neural Net), the simplest one being the MLP. As it is done in [8], MLP can be seen as a composition of n layers $h^{(i)}$, $i = 1, 2, ...n$:

The first layer is given by

$$\boldsymbol{h}^{(1)} = g^{(1)} \left(\boldsymbol{W}^{(1)\top} \boldsymbol{x} + \boldsymbol{b}^{(1)} \right)$$

and recursively

$$\boldsymbol{h}^{(i+1)} = g^{(i+!)}\left(\boldsymbol{W}^{(i+1)\top}\boldsymbol{h}^{(i)} + \boldsymbol{b}^{(i+1)}\right)$$

$\boldsymbol{W}$ is the weight matrix
$\boldsymbol{b}$ the bias vector,
g the activation function (similar to link function in Generalized Linear Models)
$\boldsymbol{x}$ is the input data.

Each layer is composed of a certain number of units which form a vector given by $\boldsymbol{W}^{(i+1)\top}\boldsymbol{h}^{(i)} + \boldsymbol{b}^{(i+1)}$.

The final layer of the network is called the output layer. We can define $h^{(0)} = x$ as the input layer. Any other layer is an hidden layer

There are many types of activation function. We define the ones we will use in this work:

$$ReLu(x) = \max(x, o)$$

$$Linear(x) = x$$

$$Sofplus(x) = \ln(1 + e^x)$$

Notice that softplus is a smoothed version of ReLu (Rectifier Linear Unit).

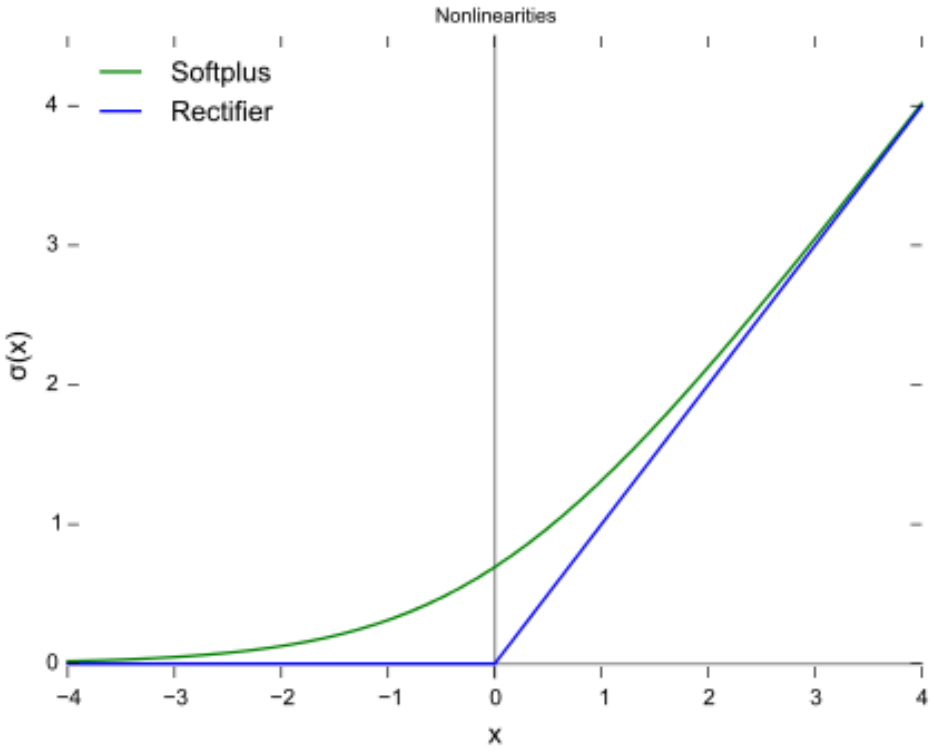


Figure 3.1 – Softplus and ReLu, taken from [**TowardsDS**]

3.4. NON PARAMETRIC GAN

The goal of the network for a supervised learning task is to approximate the function $f(x) = \mathbb{E}[Y|X = x]$ where Y is the predicted variable and X the predictive variable. The following theorem explains the powerful capacity of neural net for approximation.

Theorem 3.4.1 (Universal approximation theorem)**.** *Let I_n denote the n-dimensional unit cube, $[0,1]^n$. Let σ be bounded measurable sigmoidal function. Then finite sums of the form*

$$G(x) = \sum_{j=1}^{N} \alpha_j \sigma\left(y_j^{\mathrm{T}} x + \theta_j\right)$$

are dense in $L^1(I_n)$. In other words, given any $f \in L^1(I_n)$ and $\varepsilon > 0$, there is a sum, $G(x)$, of the above form for which

$$\|G - f\|_{L^1} = \int_{I_n} |G(x) - f(x)| dx < \varepsilon$$

Where $y_j \in \mathbb{R}^n$ and $\alpha_j, \theta \in \mathbb{R}$ are fixed.

As explained by [8], this theorem from [16] tells us that a feedforward network with a linear output layer and at least one hidden layer with a sigmoidal activation function can approximate any Borel measurable function from one finite-dimensional space to another with any desired nonzero amount of error, provided that the network is given enough hidden units. That result is also true for other activation functions.

The estimation of the neural net parameters (the weights and biases) is done by minimizing a loss unction which is generally equivalent to maximum likelihood estimation. Optimization is performed through Gradient descent and gradients are computed thanks to backpropagation which is simply an application of the chain rule for derivation through the composition of layers.

Designing and training Neural Network is an art because it relies heavily on the choice of hyperparameter. For example, one of the most important hyperparameters is the learning rate. Architecture hyperparameters include type of network architecture, number of hidden layer, number of units. Increasing the number of layers and units, improve the capacity of the network. And it is that powerful capacity that allows neural net to perform some sort of non parametric inference. But in many cases, the distribution learned by such a model is intractable. Which is why we need Kernel Density Estimation.

3.4.2 Kernel Density estimation theoretical background

Kernel density estimation (KDE) is one of the most popular non parametric density estimation method. It aims at estimating the density of $i.i.d$ samples $X_1, \ldots, X_n$ and it is defined by

$$\widehat{p}_n(x) = \frac{1}{nh} \sum_{i=1}^{n} K\left(\frac{X_i - x}{h}\right) \tag{3.9}$$

where K the kernel is a function which integrates to 1, and h is the bandwidth hyperparameter.

An example of Kernel is the Gaussian kernel. The kernel type and bandwidth are hyperparameter as they should be specified. It is generally assumed that the most important hyperparameter is the bandwidth (at least when X_i distribution is symmetric). There is a rule of thumb to choose the bandwidth designed by [20], based on the data interquantiles range and standard deviation as follow:

$$\hat{h}_{opt} = 0.9 \min\left(\hat{\sigma}; \frac{\hat{q}_3 - \hat{q}_1}{1.349}\right) n^{-\frac{1}{5}} \tag{3.10}$$

There are also methods based on cross validation to choose the optimal bandwith as explained in [6].

From [2] we have the following results concerning bias and variance for this estimator:

$$\begin{aligned}
\mathbb{E}\left(\widehat{p}_n\left(x_0\right)\right) - p\left(x_0\right) &= \int K(y) p\left(x_0 - hy\right) dy - p\left(x_0\right) \\
&= \int K(y)\left[p\left(x_0\right) + hy \cdot p'\left(x_0\right) + \frac{1}{2}h^2 y^2 p''\left(x_0\right) + o\left(h^2\right)\right] dy - p\left(x_0\right) \\
&= \int K(y) p\left(x_0\right) dy + \int K(y) hy \cdot p'\left(x_0\right) dy + \\
&\int K(y) \frac{1}{2} h^2 y^2 p''\left(x_0\right) dy + o\left(h^2\right) - p\left(x_0\right) \\
&= p\left(x_0\right) \int K(y) dy + hp'\left(x_0\right) yK(y) dy + \frac{1}{2} h^2 p''\left(x_0\right) \int y^2 K(y) dy + o\left(h^2\right) - p\left(x_0\right) \\
&= p\left(x_0\right) + \frac{1}{2} h^2 p''\left(x_0\right) \int y^2 K(y) dy - p\left(x_0\right) + o\left(h^2\right) \\
&= \frac{1}{2} h^2 p''\left(x_0\right) \int y^2 K(y) dy + o\left(h^2\right) \\
&= \frac{1}{2} h^2 p''\left(x_0\right) \mu_K + o\left(h^2\right)
\end{aligned}$$

$$\text{bias}\left(\widehat{p}_n\left(x_0\right)\right) = \frac{1}{2} h^2 p''\left(x_0\right) \mu_K + o\left(h^2\right) \tag{3.11}$$

$$
\begin{aligned}
\operatorname{Var}\left(\widehat{p}_n\left(x_0\right)\right) &= \operatorname{Var}\left(\frac{1}{nh}\sum_{i=1}^{n} K\left(\frac{X_i - x_0}{h}\right)\right) \\
&= \frac{1}{nh^2}\operatorname{Var}\left(K\left(\frac{X_i - x_0}{h}\right)\right) \\
&\leq \frac{1}{nh^2}\mathbb{E}\left(K^2\left(\frac{X_i - x_0}{h}\right)\right) \\
&= \frac{1}{nh^2}\int K^2\left(\frac{x - x_0}{h}\right) p(x)dx \\
&= \frac{1}{nh}\int K^2(y) p\left(x_0 + hy\right) dy \quad \left(\text{ using } y = \frac{x - x_0}{h} \text{ and } dy = dx/h \text{ again }\right) \\
&= \frac{1}{nh}\int K^2(y)\left[p\left(x_0\right) + \mathrm{hyp}'\left(x_0\right) + o(h)\right] dy \\
&= \frac{1}{nh}\left(p\left(x_0\right)\cdot\int K^2(y)dy + o(h)\right) \\
&= \frac{1}{nh}p\left(x_0\right)\int K^2(y)dy + o\left(\frac{1}{nh}\right) \\
&= \frac{1}{nh}p\left(x_0\right)\sigma_K^2 + o\left(\frac{1}{nh}\right)
\end{aligned}
$$

Where

$$\mu_K = \int y^2 K(y)dy \tag{3.12}$$

$$\sigma_K^2 = \int K^2(y)dy \tag{3.13}$$

3.4.3 Non parametric GAN Estimation

We want to resolve the objective optimization problem in the case where G is given by a MLP. In the case where G generates a continuous variable, there is no method to compute an analytical form of p_g. Therefore we estimate it with KDE. Our estimator is then the solution of

$$
\begin{cases}
\max\limits_{cte} \dfrac{1}{n}\sum\limits_{x} log(\dfrac{p_m(x)}{cte p_m(x) + \hat{p}_g(x)}) + \dfrac{1}{n}\sum\limits_{y} log(\dfrac{\hat{p}_g(y)}{cte p_m(y) + \hat{p}_g(y)}) \\
\min\limits_{G} \dfrac{1}{n}\sum\limits_{y} log(\dfrac{\hat{p}_g(y)}{cte p_m(y) + \hat{p}_g(y)})
\end{cases}
$$

where $\hat{p}_g$ is given by a KDE with a Gaussian Kernel, with a manually tuned bandwith. We consider the estimation of *cte* for normal distribution and exponential distribution.

Since the structure of our data is fairly simple, we are going to use a simple architecture for our neural network. We use a MLP with one hidden layer with 15 units and Relu

activation function for the hidden layer. Since the output of our MLP is not bounded, we use a linear activation function for the Gaussian case. For the exponential case, we use ReLu (We also used Softplus, with similar results), which gives better result than Relu. We choose a 5 dimensional standard normal latent variable z, as input for the generator. This model architecture is inspired by [3]. We provide here an illustration of G's architecture :

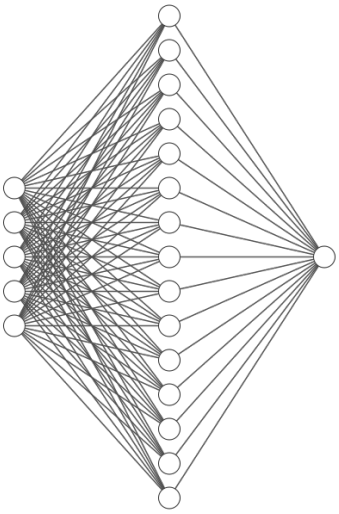


Figure 3.2 – Generator Architecture, Figure produced with alexnail.me

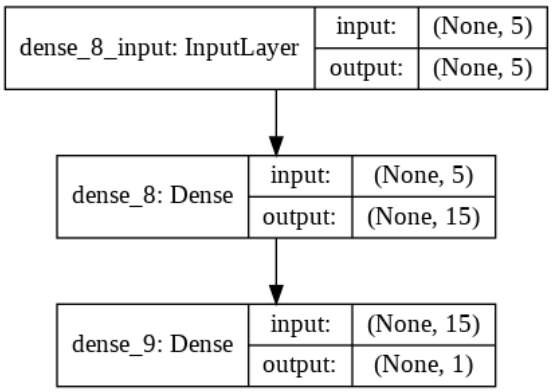


Figure 3.3 – Generator Architecture obtained with keras

We adapt Algorithm 1 to our setting, by incorporating a step for KDE update:

3.4. NON PARAMETRIC GAN

Algorithm 2 Minibatch stochastic gradient descent training of adversarial contrastive estimator

for nb training iterations **do**
  for k steps **do**
    Sample minibatch of m noise samples $\{z_{(1)}, ..., z_{(m)}\}$ fromp_z;
    Sample minibatch of m examples $\{x_{(1)}, ..., x_{(m)}\}$ from p_z;
    Estimate $\hat{p}_g$ with KDE
    Update the discriminator by ascending its stochastic gradient:

$$\nabla_{cte} \frac{1}{n} \sum_x log(\frac{p_m(x)}{cte p_m(x) + \hat{p}_g(x)}) + \frac{1}{n} \sum_y log \frac{\hat{p}_g(y)}{cte p_m(y) + \hat{p}_g(y)})$$

  end for
  Sample minibatch of m noise samples $\{z_{(1)}, ..., z_{(m)}\}$ from p_z
  Estimate $\hat{p}_g$ with KDE
  Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{n} \sum_y log \frac{\hat{p}_g(y)}{cte p_m(y) + \hat{p}_g(y)})$$

end for

We train our model according to Algorithm 2. We use Tensorflow and keras libraries for model architecture. We implement the NCE discriminator as a custom loss function. We use the Adam [15] optimizer for the neural net and a handcrafted gradient descent for the discriminator optimization. In the case of the Exponential distribution, a manually tuned decreasing learning rate schedule was needed. We implement the KDE with Tensorflow Probability module.

We provide here numerical results of our model estimate for several data parameters

Data	True cte	$\hat{cte}$ estimate	G mean	G std dev
$x \sim N(3, 0.2)$	1.99	1.91	3.04	0.4
$x \sim E(rate = 0.5)$	0.5	0.43	1,8	2.04

We can see that G's output converge to the data distribution :

3.4. NON PARAMETRIC GAN

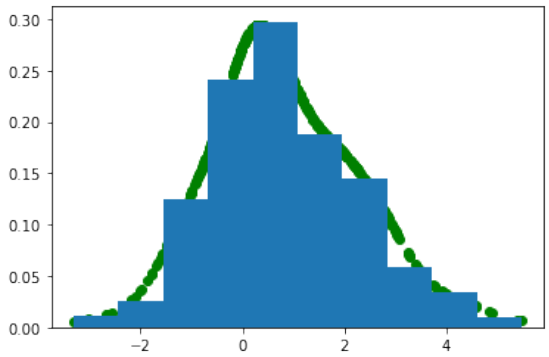


Figure 3.4 – G output for the Gaussian at iteration 0 : $\mu_g = 0.83$ et $\sigma_g = 1.45$, KDE in green

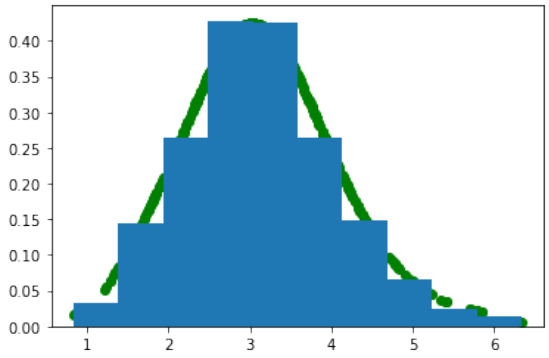


Figure 3.5 – G output for the Gaussian at mid iteration : $\mu_g = 3.05$ et $\sigma_g = 1.22$, KDE in green

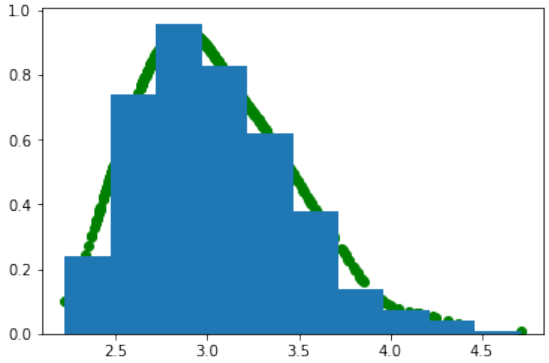


Figure 3.6 – G output for the Gaussian at final iteration : $\mu_g = 3$ et $\sigma_g = 0.5$, KDE in green

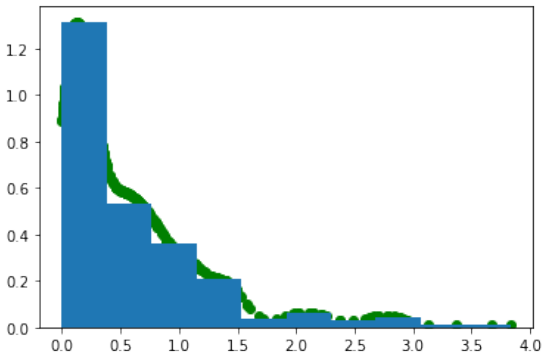


Figure 3.7 – G output for the Exp at first iteration : $\mu_g = 0.61$ et $\sigma_g = 0.12$, KDE in green

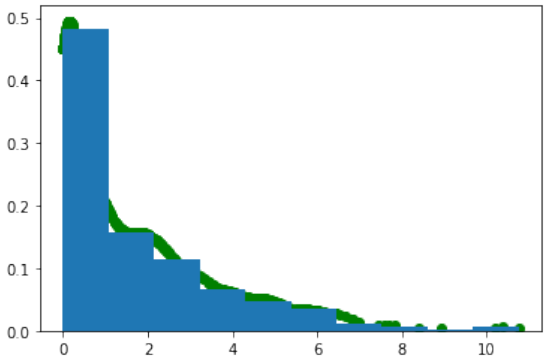


Figure 3.8 – G output for the Exp at mid iteration : $\mu_g = 1.42$ et $\sigma_g = 1.53$, KDE in green

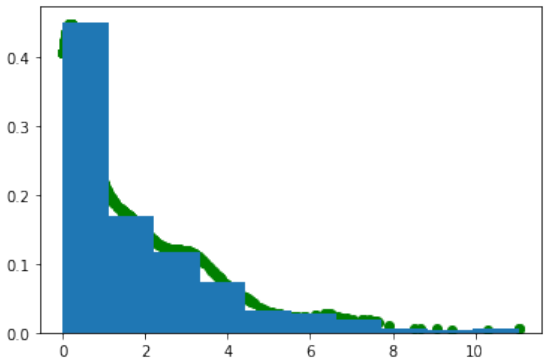


Figure 3.9 – G output for the Exp at last iteration : $\mu_g = 1.80$ et $\sigma_g = 2.04$, KDE in green

3.4. NON PARAMETRIC GAN

Discussion : It is worth noting that the quality of estimation and convergence is heavily crippled by the quality of KDE. It is more difficult to perform KDE for the Exponential since it is an asymmetric distribution. Some KDE kernels have been designed for this [6]. But still, using this method in high dimension would not give satisfying result since KDE does not work well in high dimension.

3.5 GAN vs NCE

We provide here some experimental results which show that ACE performs almost as well as NCE with a proper choice of noise. The estimate are average over 100 runs for a Gaussian $N(\mu = 0.5, \sigma = 0.2)$.

Data	True cte	$\hat{cte}$ estimate	MSE
NCE $\mu = 2.5$ $\sigma = 0.4$	1.994	4.63	58.72
NCE $p_n = p_m$	1.994	1.994	2.41e-30
NCE $\mu = 1$ $\sigma = 0.4$	1.994	1.996	0.0051
ACE	1.998	1.994	1.894e-05

Chapter 4

Conclusion

NCE is an effective method to estimate the normalizing factor of an unnormalized probability density. Its main drawback is the fact that it relies on a proper choice of the noise distribution, with no clear heuristic to decide if a specific noise is good enough or not. From intuition, we know that the closer the noise is to the data, the more precise the estimation will be, even if we don't have strong theoretical evidence of it. Adversarial Contrastive Estimation is therefore a promising method to overcome the noise choice problem. Indeed, we showed that ACE perform almost as well as NCE with the proper choice of noise. The main limit of our approach is then the fact that we need a way to estimate the density of the generator when it is a neural network. KDE can be useful in one dimension but not in the multidimensional case. Future work could therefore explore the use of Masked Autoregressive models [18], which seem to be a promising approach to build a neural generative model with a robust estimation method.

Appendix A

Appendix NCE : Figures 6, 7 and 8

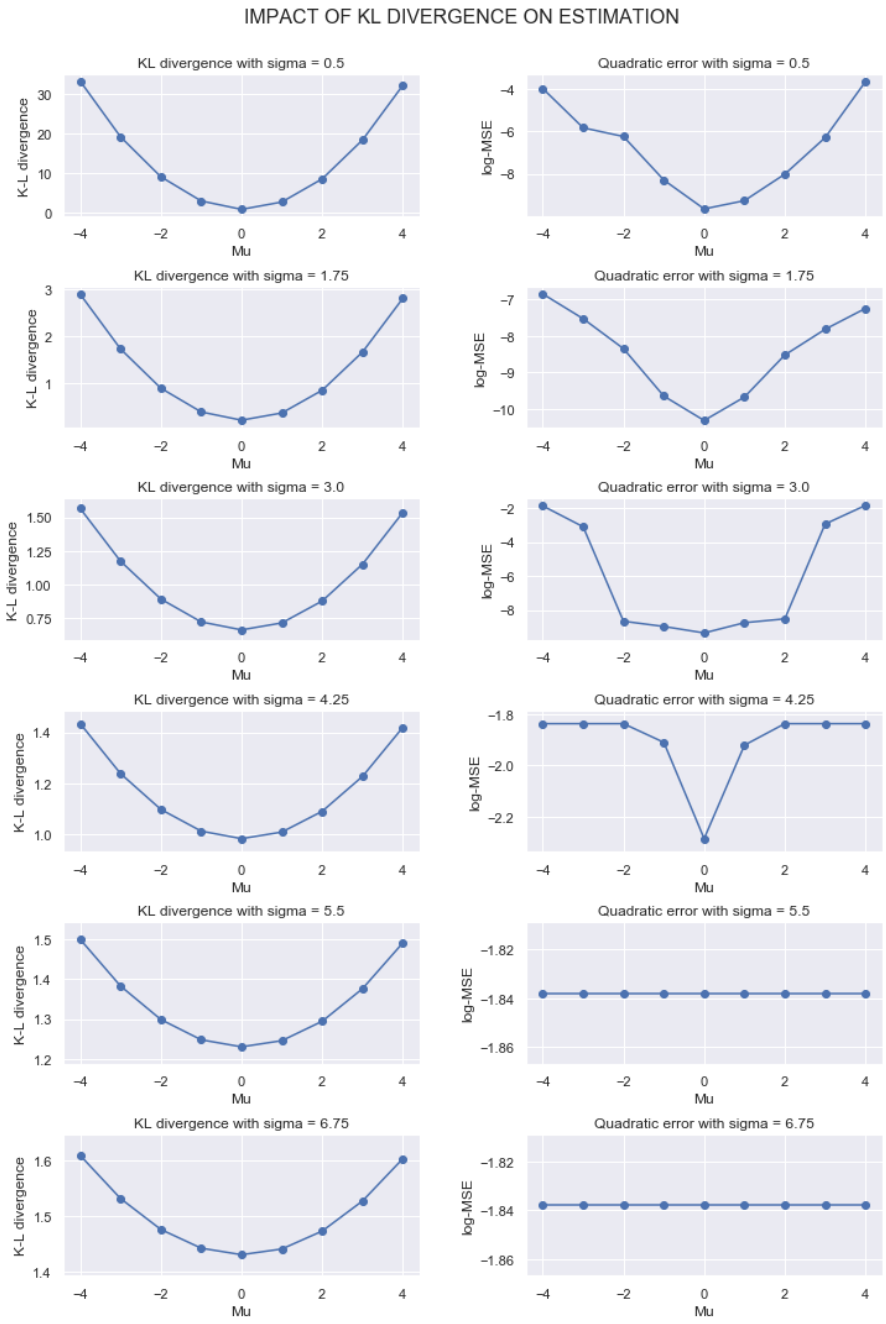


Figure A.1 – Kullback-Liebler divergence "limits" with respect to mu and sigma fixed

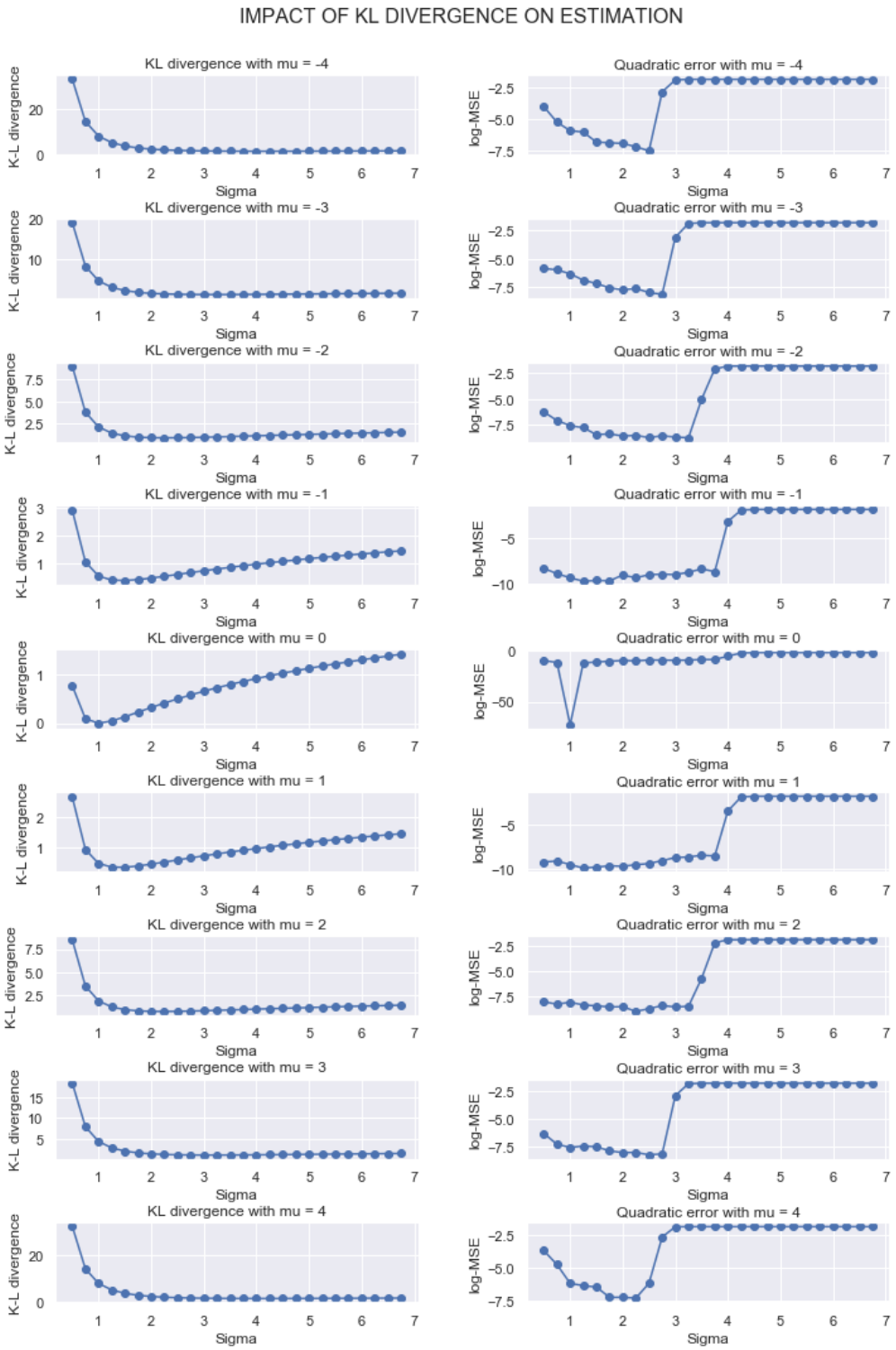


Figure A.2 – Kullback-Liebler divergence "limits"respect to sigma and mu fixed

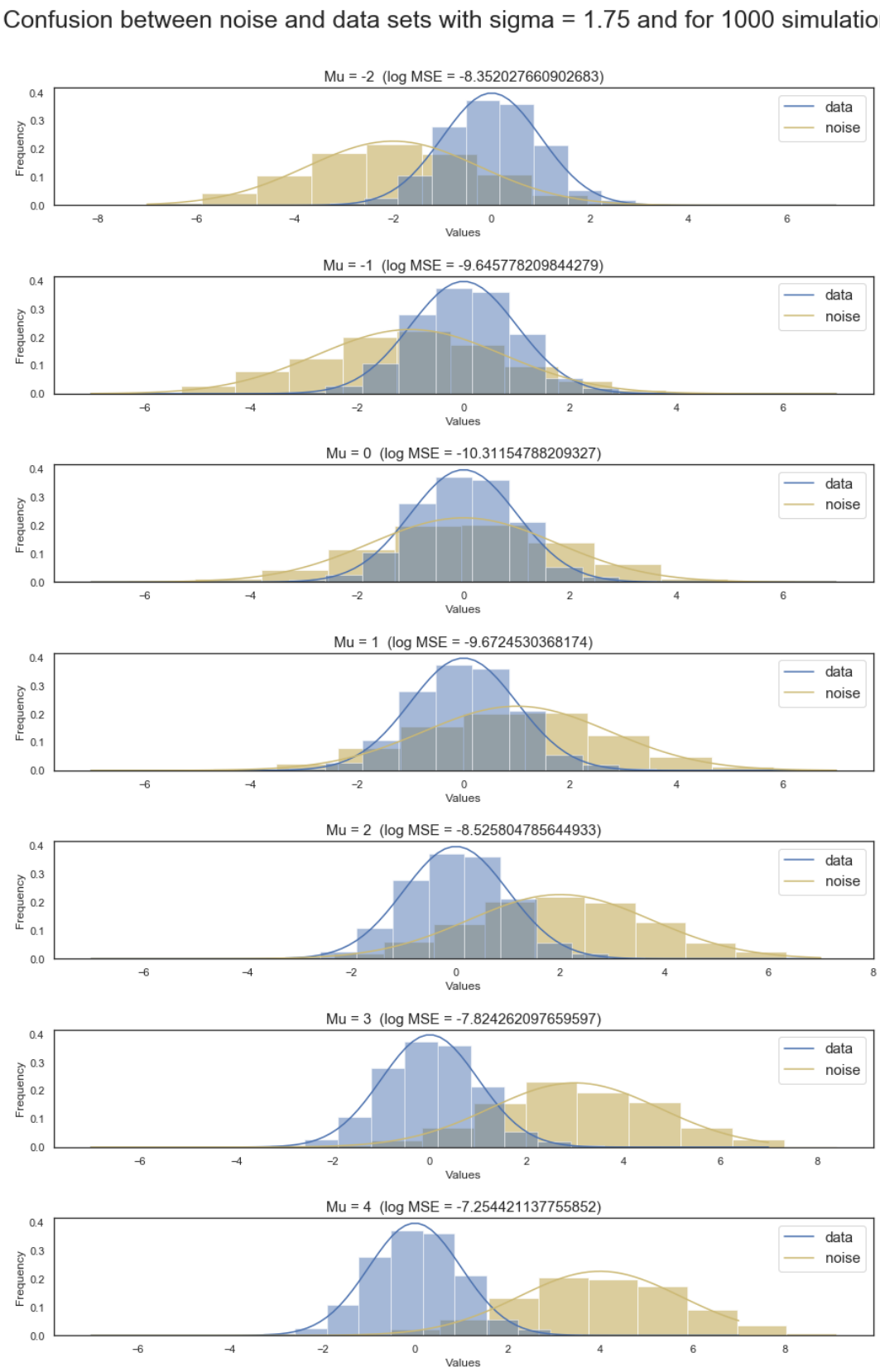


Figure A.3 – Confusion between data and noise samples with respect to mu's value when $\sigma = 1.75$: We can see the histogram to be the most confused when $\mu_{data} = \mu_{noise} = 0$. This coincides with the point where the Log(MSE) reaches its minimum.

Bibliography

[1] https://ziqiaowanggeothe.github.io/2018/10/01/Comparison-between-NCE-and-GAN/.

[2] http://faculty.washington.edu/yenchic/17Sp_403/Lec7-density.pdf.

[3] https://machinelearningmastery.com/how-to-develop-a-generative-adversarial-network-for-a-1-dimensional-function-from-scratch-in-keras/.

[4] Julian Besag. "Spatial Interaction and the Statistical Analysis of Lattice Systems". In: *Journal of the Royal Statistical Society. Series B (Methodological)* 36.2 (1974), pp. 192–236. ISSN: 00359246. URL: http://www.jstor.org/stable/2984812.

[5] Avishek Joey Bose, Huan Ling, and Yanshuai Cao. "Adversarial Contrastive Estimation". In: *arXiv e-prints*, arXiv:1805.03642 (May 2018), arXiv:1805.03642. arXiv: 1805.03642 [cs.CL].

[6] Arthur Charpentier and Emmanuel Flachaire. "Log-Transform Kernel Density Estimation of Income Distribution". In: *SSRN Electronic Journal* (Mar. 2015). DOI: 10.2139/ssrn.2514882.

[7] Ian J. Goodfellow et al. "Generative Adversarial Networks". In: *arXiv e-prints*, arXiv:1406.2661 (June 2014), arXiv:1406.2661. arXiv: 1406.2661 [stat.ML].

[8] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[9] Ian Goodfellow et al. "Generative Adversarial Networks". In: *Advances in Neural Information Processing Systems* 3 (June 2014).

[10] Michael Gutmann and Aapo Hyvärinen. "Estimation of unnormalized statistical models without numerical integration". In: *Proc. Workshop on Information Theoretic Methods in Science and Engineering (WITMSE2013)*. 2013.

[11] Michael Gutmann and Aapo Hyvärinen. "Noise-Contrastive Estimation of Unnormalized Statistical Models, with Applications to Natural Image Statistics". In: *Journal of Machine Learning Research* 13 (Feb. 2012), pp. 307–361.

[12] Geoffrey E. Hinton. "Training Products of Experts by Minimizing Contrastive Divergence". In: *Neural Computation* 14.8 (2002), pp. 1771–1800. DOI: 10.1162/089976602760128018, eprint: https://doi.org/10.1162/089976602760128018. URL: https://doi.org/10.1162/089976602760128018.

[13] Aapo Hyvärinen. "Estimation of Non-Normalized Statistical Models by Score Matching." In: *Journal of Machine Learning Research* 6 (Jan. 2005), pp. 695–709.

[14] Diederik P Kingma and Max Welling. *Auto-Encoding Variational Bayes*. 2013. arXiv: 1312.6114 [stat.ML].

[15] Diederik Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *International Conference on Learning Representations* (Dec. 2014).

[16] Grzegorz Lewicki and Giuseppe Marino. "Approximation by Superpositions of a Sigmoidal Function". In: *Appl. Math. Lett.* 17 (Dec. 2004), pp. 1147–1152. DOI: 10.1016/j.aml.2003.11.006.

[17] Andriy Mnih and Yee Whye Teh. "A Fast and Simple Algorithm for Training Neural Probabilistic Language Models". In: *arXiv e-prints*, arXiv:1206.6426 (June 2012), arXiv:1206.6426. arXiv: 1206.6426 [cs.CL].

[18] George Papamakarios, Theo Pavlakou, and Iain Murray. *Masked Autoregressive Flow for Density Estimation*. 2017. arXiv: 1705.07057 [stat.ML].

[19] Christian Robert. "The Bayesian Choice: From Decision Theoretic Foundations to Computational Implementation". In: (Jan. 2007).

[20] B.W. Silverman. "Density Estimation for Statistics and Data Analysis". In: (1986). URL: %5Curl%7Bhttps://ned.ipac.caltech.edu/level5/March02/Silverman/paper.pdf%7D.