UNIVERSIDADE ESTADUAL DE CAMPINAS
INSTITUTO DE MATEMÁTICA, ESTATÍSTICA E COMPUTAÇÃO CIENTÍFICA
DEPARTAMENTO DE MATEMÁTICA APLICADA

Igor Cardoso Rosica

Representation Learning: A starting point for probabilistic models.

Campinas
21/08/2020

Igor Cardoso Rosica

Representation Learning: A starting point for probabilistic models.*

Monografia apresentada ao Instituto de Matemática, Estatística e Computação Científica da Universidade Estadual de Campinas como parte dos requisitos para obtenção de créditos na disciplina Projeto Supervisionado, sob a orientação do(a) Prof. Adín Ramírez Rivera.

*Este trabalho foi financiado pelo Governo do Estado de São Paulo, Brasil.

Abstract

Looking forward to be an introductory text to probabilistic models in *Representation Learning*, this work has two main related purposes. The first one is to contribute with the undergraduate community of students that are looking for a first contact with the theme, while the second is regarded to the intention of the author to approach this universe. For that, we bring here an expository work, followed sometimes of informal language, about topics that are, in many cases, strongly related with our main theme. Between these topics are some of the main problematics within the machine learning field, what are Boltzmann machines and how they relate with the representation learning paradigm, the characterization of some standard probabilistic models, graphical models and how to deal with intractable distributions. By the end of the work we present some conclusion and motivations obtained during the process of writing.

Resumo

Em busca de ser um texto introdutório ao assunto de modelos probabilísticos em *Aprendizado de Representações* este trabalho possui dois propósitos principais relacionados. O primeiro é contrubuir com a comunidade de alunos de graduação que buscam um primeiro contato com o assunto, enquanto o segundo é resumido pela intenção do autor de se aproximar deste universo. Para isso, trazemos aqui um trabalho expositivo seguido de explicações descontraídas sobre tópicos que estão, em algumas vezes, fortemente relacionados ao nosso tema principal. Entre estes tópicos abordados estão algumas das principais problemáticas envolvidas no contexto de *Machine Learning*, o que são maquinas de Boltzmann e como elas se relacionam com a história do campo de *Representation Learning*, a caracterização de alguns modelos probabilíscos convencionais e bayesianos, modelos gráficos e como lidar com distribuições intratáveis utilizando inferência variacional. Ao final apresentamos uma discussão sobre conclusões e motivações obtidas durante o processo de escrita.

Contents

1 Introduction **6**

1.1 But how this really works? 6

1.2 The problems of dimensionality and number of variations 7

1.3 Distributed representations and multi-clustering 8

1.4 How do we approach the R.L. field? 8

2 Single-Layer, Depth and the Underlying Structure of an Environment **9**

2.1 Standard Feedforward Network and SGD 9

2.2 One step back 12

2.2.1 Boltzmann Machines 12

2.2.2 Belief Nets 15

2.2.3 Greedy Layer-Wise Learning 16

2.3 Depth 17

3 Probabilistic Approach **18**

3.1 What is a Bayesian Approach? 18

3.2 Graphical Models 20

3.2.1 Directed Graphical Models 21

3.2.2 Undirected Graphical Models 22

3.3 Dealing with Intractable Posteriors 25

3.3.1 Variational Inference and the Mean Field Method 25

4 Conclusions and Motivations **27**

1 Introduction

Machine learning techniques became a common tool between the data scientists. Because of its large spectrum of applications on industry this theme has taken a lot of space in the media. On the other side, we can see that the modern Artificial Intelligence subject and its derivatives is a quite old fashioned content in the tech industry and also in the academic environment, furthermore in this context both are strongly connected. From this point of view we can see that: "The performance of machine learning methods is heavily dependent on the choice of data representation (or features) on which they are applied", [Bengio et al., 2013]. In a simple manner, the *Representation Learning* field is the study of how to learn representations that compress the information of a generic data set, in a way of optimizing a given task. During the past decades the concepts within *Representation Learning (R.L.)* field are strongly connected with the ascension of *Deep Architectures*, as we will see in Section 2, and these ideas have been widely used to successfully improve many areas of machine learning, including speech recognition, signal processing, computer vision, natural language processing and also transfer learning, which stands for "the ability of a learning algorithm to exploit commonalities between different learning tasks in order to share statistical strength, and transfer knowledge across tasks" [Bengio et al., 2013].

1.1 But how this really works?

To clarify a little our hitchhiker's mind, including myself, let's first understand how a standard machine learning model works, to do so consider the following statement: The simplest idea here is that machine learning models are concerned about automating methods of data analysis, and this can be split primarily on two subsets. The first one is called predictive or ***supervised learning***, which the key goal is to obtain a mapping function from inputs $\mathbf{x}$ to outputs y, using a given labeled *training set* $\vec{D} = \{(x_i, y_i)\}_{i=1}^{N}$, where N is the number of training examples. We call our inputs $\mathbf{x}$ and this could be a vector or whatever other relevant math element, such as a tensor. It is important to notice that, our inputs are not the features of data itself, or real qualities of our observed object, like height or weight, but a representation of it and this is one of the motivations of our

theme of study. It is quite known that data hides in itself a lot of uncovered information, like for example unknown dimensions and manifolds, this hidden information is what we are interested in our work. The output element, by its turn, y is called our *response variable* and it could, in principle, be anything but we can also, by simplicity, split in two types, the first one is called classification and occurs when y_i is a *categorical* variable from some finite set, $y_i \in \{1, ..., C\}$, the second one is called *regression* and occurs when y_i is a real valued variable. The second one of the machine learning models that we are talking is the ***unsupervised learning*** kind, this type of models are characterized by having a *training set* $D = \{x_i\}_{i=1}^{N}$ composed by only inputs, where the main intent is to find information related to not labeled data, like interesting patterns as clusters or anomaly detection. Finally, to complete our initial exposure, there is another popular kind of machine learning models, and it is named *reinforcement learning*, which stands for how a given *agent* would take some *action* or *behave* when given occasional reward or punishment.

For most methods exposed above and for many problems in this context exists a well defined task or objective, like obtaining a successful classifier or predictor for a given dataset. By the way, the theme we are are about to study is quite different, mainly because: "One of the challenges of representation learning that distinguishes it from other machine learning tasks such as classification is the difficulty in establishing a clear objective, or target for training. In the case of classification, the objective is (at least conceptually) obvious, we want to minimize the number of misclassifications on the training dataset. In the case of representation learning, our objective is far-removed from the ultimate objective, which is typically learning a classifier or some other predictor" [Bengio et al., 2013].

1.2 The problems of dimensionality and number of variations

An old difficult fact of machine learning field is that as many as the number of features, or dimensions, grows so the amount of data we need to generalize accurately grows exponentially, or in a better way we can say that a local generalization depends directly on representative examples for all relevant variations, this problem is known as *the curse of dimensionality*. Many techniques, such as linear regression and standard

kernel machines, relies on a generalization based on some kind of interpolation between neighbors from the data set, that is strongly related with the *smoothness prior*, i.e., assuming that our wanted target function is smooth enough as the statement $x_1 \approx x_2 \implies f(x_1) \approx f(x_2)$. This is a good but insufficient prior, since real data generally stands on a non conventional and highly curved manifold, where the number of variations may grow exponentially with the number of factors which causes changes on the raw input representation, so the search for methods that are not only concerned on the smoothness prior are really valid in this context.

1.3 Distributed representations and multi-clustering

Simple clustering, nearest neighbors or even decision trees algorithms can be simplified as breaking the input space in regions, so you have some kind of behavior on each one of these regions and then looking for a manner to refine the mutually exclusive clusters that you have constructed. The problem with these approaches is that the number of distinguishable regions grows linearly with the number of parameters involved. By the other side, there is another family of algorithms called *multi-clustering*, such as *Neural Nets*, where the same clustering is applied on different parts of the input. "In a distributed representation, an exponentially large number of possible subsets of features or hidden units can be activated in response to a given input. In a single layer model, each feature is typically associated with a preferred input direction, corresponding to a hyperplane in input space, and the code or representation associated with that input is precisely the pattern of activation" [Bengio et al., 2013]. The idea behind the importance of distributed representations is that by using them it is possible to generalize non locally to never seen regions, that is exploring the fact each feature has a global relevance in your model.

1.4 How do we approach the R.L. field?

There are many different approaches that are concerned about capturing posterior beliefs from raw data, for most of these techniques the idea behind it is to find manners to automate the extraction of features, in this work we will try to focus on a little part of the probabilistic approaches. "Since the dataset is itself a random variable,

the learning process involves the application of a procedure to a target distribution from which the examples are drawn and for which one would like to infer a good decision function" [Bengio and Delalleau, 2011]. The formalism that we will focus here stands for constructing posterior distributions using the joint of *latent variables* $\mathbf{h}$ and data $\mathbf{x}$, and then, by maximizing likelihood, possible obtaining a representation $\mathbf{h}$ of $\mathbf{x}$.

2 Single-Layer, Depth and the Underlying Structure of an Environment

Before we set what is our *latent variables* and how we will define our observation data, let's first see one of the reasons that motivated all the representation learning paradigm. As we mentioned in our first section, the artificial intelligence subject is a really old fashioned content in the academy, so as the idea of *backpropagation* and *Belief Nets*. In this section we expect to give a high overview of these simple ideas and how that motivated the R.L. field, passing through the famous *single-layer greedy learning modules* [Hinton et al., 2006].

2.1 Standard Feedforward Network and SGD

Consider a simple example of a supervised learning model such as *linear regression*, as we mentioned before the objective of these methods is to obtain a mapping function using labeled inputs, in a manner of automating a given task, which could be a classification for example. To do so, we must decide how the *target function* that we are looking to find looks like, and how to get as close as possible to it by adjusting parameters, i.e., a construction of ideas concerned in resulting a solution for our task.

Suppose that our training examples are given by $\{(x^{(i)}, y^{(i)})\}_{i=1}^{N}$, where our expected outputs are $y_i \in \mathbb{R}$, and our target function $h_\theta(x) \in \mathbb{R}$ is defined in terms of the parameters θ. A standard approach, in this case, is to use a *cost function*. The cost function here is a function that maps an event or values of one or more variables onto a real number representing some cost associated with the event, many optimization models are related with the minimization of it. Consider a case that our cost function is the *least*

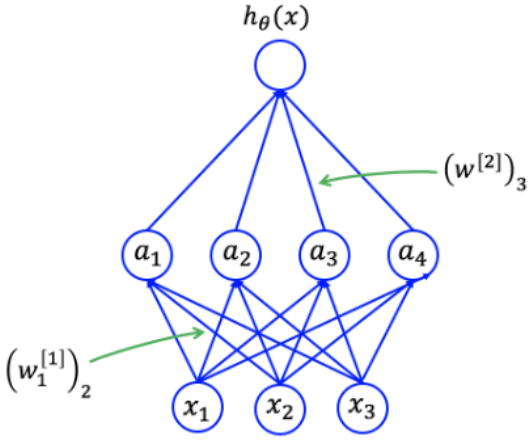

Figure 1: Example of a fully connected neural network, extracted from Goodfellow et al. [2016]

squared cost function defined for each example $(x^{(i)}, y^{(i)})$ as

$$J^{(i)}(\theta) = \frac{1}{2}(h_\theta(x^{(i)}) - y^{(i)})^2, \tag{1}$$

and the *mean-square cost function* for the entire dataset is consequently defined as

$$J(\theta) = \frac{1}{n}\sum_{i=1}^{n} J^{(i)}(\theta). \tag{2}$$

Now consider the simple example of a *fully-connected neural network* with three layers, the input layer, the *hidden layer* and the output layer. The simplest idea to construct here is that we can relate our inputs x_i to our outputs $h_\theta(x)$ with a composition of functions using weights w, biases b and activations through hidden units a. Each edge has a unique weight, and each node has a unique bias. The composition works as the following way, a two-layer fully-connected neural network with m hidden units and a d

dimensional input $x \in \mathbb{R}$ is defined as

$$\forall j \in [1, ..., m], \quad z_j = w_j^{[1]^T} + b_j^{[1]}, \quad \text{where } w_j^{[1]} \in \mathbb{R}^d, b_j^{[1]} \in \mathbb{R} \tag{3}$$

$$a_j = \text{ReLU}(z_j) \tag{4}$$

$$a = [a_1, ..., a_m]^T \in \mathbb{R}^m \tag{5}$$

$$h_\theta(x) = w^{[2]^T} a + b^{[2]}, \quad \text{where } w^{[2]} \in \mathbb{R}^m, b^{[2]} \in \mathbb{R} \tag{6}$$

This treatment makes the relation between our input and output pretty clear through the entire net. We can say that we know how everything is connected on each node. In matrix notation we have that

$$z = W^{[1]} x + b^{[1]} \tag{7}$$

$$a = \text{ReLU}(z) \tag{8}$$

$$h_\theta(x) = W^{[2]} a + b^{[2]}. \tag{9}$$

This approach can be easily extended to as many layers as you want, that is, as many parameters and compositions as you want, think about how general this could be. If you look to our cost function $J^{(1)}(\theta)$ on the equation (2), you can see that it depends only on h_θ and the labels y. By the method we constructed the network, having on each node a composition of the previous ones and using a ReLU activation, it is possible, in many cases, to easily take the derivatives of our cost function with respect to every single weight using the *chain rule*. This idea of *backpropagation* makes possible to compare our output $h_\theta(x)$ with the labeled expected values y, and so update our weights to approximate the desirable function, but only after rolling out a bunch of examples from the training set. One of these approaches that is fairly known is the *stochastic gradient descent*, the idea behind it is pretty simple and can be resumed as the following.

Looking forward to optimize a parameter $\vec{\theta}$, for a given *learning rate* α, we do, for each example of the training set (x_i, y_i), the update:

$$\theta_j = \theta_j - \alpha \nabla_{\theta_j} J^{(i)}(\theta_j) \tag{10}$$

2.2 One step back

What we want to expose here is that these ideas about neural nets are a really old content, and has taken a lot of neurons from the scientists through the decades. Think about it as if you were a geek living in the 80's playing your video game. Atari looks pretty better, right? Depending on your network it could take a lot of computational cost. Although, often, you may need so much labeled data, such as a good initialization for your weights. For these reasons and many others that's when the *probabilistic models* have started to gain more attention. At the same time that these ideas were taking neurons from the geeks, there was an approximation between physics and networks in this context going on, which later will result on a really important algorithm.

While neural nets were falling down in the 80's, the scientists were looking for manners to overcome the limitations of it. One of the ways that they have taken was the *generative models*, i.e., forms to model the input data rather than predicting a label. In this context, we want to clarify two important trends that were going on during the 80's and early 90's, *Boltzmann Machines* and *Belief Nets*.

2.2.1 Boltzmann Machines

As we will see here, a Boltzmann machine is a really elegant manner for obtaining distributions that represents binary vectors, and has a learning algorithm so much simple. To construct a Boltzmann machine take some fixed number of binary units linked by symmetrical connections. The state of a unit i is given by s_i, while the state vector of our entire network is defined as $\mathbf{s}$. We can think this state vector as a collapse of a given random variable S. The weights between two generic units i and j are given by $w_{ij} = w_{ji}$. So the energy of the entire network, with biases b_i is

$$E(\mathbf{s}) = -\sum_i s_i b_i - \sum_{j<i} s_i s_j w_{ij}. \tag{11}$$

The energy gap ΔE_i of a unit s_i represents the difference of a unit in the total energy of our network

$$\Delta E_i = E(s_i = 0) - E(s_i = 1) = b_i + \sum_j s_j w_{ij}. \tag{12}$$

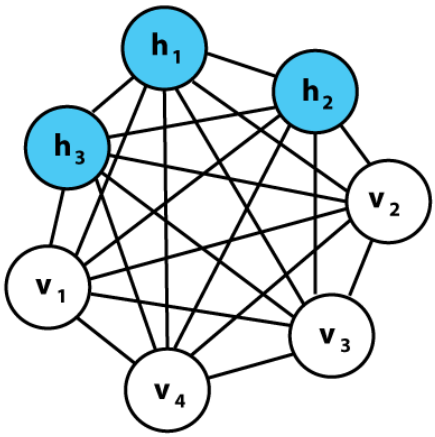

Figure 2: Example of Boltzmann Machine, Public Domain.

Our typical approach has *hidden units* h_k and *visible units* v_i. So, the joint total energy becomes

$$E(\mathbf{v}, \mathbf{h}; \theta) = -\sum_i v_i b_i - \sum_k h_k b_k - \sum_{i<j} v_i v_j w_{ij} - \sum_{i,k} v_i h_k w_{ik} - \sum_{k<l} h_k h_l w_{kl}, \tag{13}$$

where θ is our parameter object, which in this case are the biases and weights.

Now, we replace our binary threshold units by *binary stochastic units*, which makes biased random decisions based on a Boltzmann distribution and our temperature parameter T

$$p(s_i = 1) = \frac{1}{1 + e^{-\Delta E_i / T}}. \tag{14}$$

This step allows us to think in our units as "*particles*". For simplicity, let's take $T = 1$. Then, after a given time, the joint probability distribution takes the form

$$p(\mathbf{v}, \mathbf{h}; \theta) = \frac{e^{E(\mathbf{v}, \mathbf{h}, \theta)}}{Z}, \tag{15}$$

where $Z(\theta) = \sum_{\mathbf{h},\mathbf{v}} e^{-E(\mathbf{v}, \mathbf{h}, \theta)}$ is the partition function, that is the sum over all possible configurations. The marginal distributions of the units will be given by

$$p(\mathbf{v}; \theta) = \sum_{\mathbf{h}} p(\mathbf{v}, \mathbf{h}; \theta) = \frac{1}{Z(\theta)} \sum_{\mathbf{h}} e^{-E(\mathbf{v}, \mathbf{h}; \theta)} \tag{16}$$

$$p(\mathbf{h};\theta) = \sum_{\mathbf{v}} p(\mathbf{v},\mathbf{h};\theta) = \frac{1}{Z(\theta)} \sum_{\mathbf{v}} e^{-E(\mathbf{v},\mathbf{h};\theta)}. \tag{17}$$

Now consider the energy of the system with a visible vector $\mathbf{v}$ *clamped*, i.e., fixed

$$E(\mathbf{v}) = -\sum_{i} \mathbf{s}_i^{\mathbf{v}} b_i - \sum_{i<j} \mathbf{s}_i^{\mathbf{v}} \mathbf{s}_j^{\mathbf{v}} w_{ij}, \tag{18}$$

where $\mathbf{s}_i^{\mathbf{v}}$ is the binary state assigned to unit i by the visible vector $\mathbf{v}$. Then,

$$\frac{\partial E(\mathbf{v})}{\partial w_{ij}} = -\mathbf{s}_i^{\mathbf{v}} \mathbf{s}_j^{\mathbf{v}}. \tag{19}$$

On the other hand, without *clamping* the system, with the stochastic choice that we made in (14), after a given time, with the system in *thermal equilibrium*, the probability of the state given by our visible vector $\mathbf{v}$ is

$$P(\mathbf{v}) = \frac{e^{-E(\mathbf{v})}}{Z}. \tag{20}$$

So the beauty of the work made by David H. Ackley [1985] is that by taking the derivative of the log of this equation, and using the result (19), would lead us to the following:

$$\sum_{\mathbf{v}\in \text{data}} \frac{\partial \log P(\mathbf{v})}{\partial w_{ij}} = < \mathbf{s}_i \mathbf{s}_j >_{\text{data}} - < \mathbf{s}_i \mathbf{s}_j >_{\text{model}}, \tag{21}$$

where $< \mathbf{s}_i \mathbf{s}_j >_{\text{data}}$ is the expected value of $\mathbf{s}_i \mathbf{s}_j$ with $\mathbf{v}$ clamped and $< \mathbf{s}_i \mathbf{s}_j >_{\text{model}}$ is the expected value when the Boltzmann machine is sampling back state vectors without clamping the system. **This result is one of the alternatives to backpropagation that the scientists were looking for during the 80's**. With this derivative is possible to create a *gradient ascent* relation for training.

The training procedure is regarded to obtain the parameters θ that makes the marginal probability over the visible units $p(\mathbf{v},\theta)$ as close as possible to the unknown probability $p_{data}(\mathbf{v})$. This can be reached, using an identically independent sampled training set $\vec{D}$ of visible vectors, by maximizing the log likelihood of the parameters $\mathcal{L}(\theta|\vec{D}) = \sum_{i=1}^{N} \log p(\mathbf{v}^{(i)};\theta)$, and this is equivalent to minimize the *Kullback-Leibler* di-

vergence or the relative entropy of $p(\mathbf{v};\theta)$ from the unknown $q(\mathbf{v})$

$$D_{KL}(q||p) = \sum_{\mathbf{v}} q(\mathbf{v}) \log \frac{q(\mathbf{v})}{p(\mathbf{v};\theta)}. \tag{22}$$

There is an interesting idea here. It is possible to model the underlying structure of an environment, by making configurations with our visible units! But the question is: How do we get the statistics for $p(\mathbf{v};\theta)$? The first barrier to do this is to obtain $Z(\theta)$, and the second is to marginalize $p(\mathbf{v},\mathbf{h};\theta)$. If our network has too many hidden units this will lead us to intractable distribution. One of the alternatives, that also was used in those times, is *sampling*.

2.2.2 Belief Nets

During the the same period that the physicists were helping the DOS guys, there was another kind of "miraculous" nets hyping through the labs. As you can imagine, the Boltzmann machines (B.M.'s) also were not the solution for all the problems in the world. One of the bad things is that, besides the slowness, they needed a negative phase for learning, yes a negative phase on the *clamping* thing! Between the means to avoid this variety of problems it was the other main side of the generative models, the *Belief Networks*, which were based in *causal relations* rather than an energy approach. As mentioned in [Neal, 1992], these networks were early related with the purpose of representing knowledge obtained from human experts. To better represent this knowledge, the nets with this purpose were directed, i.e., flows in a direction with no symmetric weights, and had also stochastic hidden units, so as the the connection between the nodes represented the causes, such as probabilities that was fairly known by an *expert*.

An important property of this family of nets is that the causal relations has as its consequence the *explaining away* effect. The inference paradigm can be understood as how to infer the state of the hidden units, while the learning procedure is regarded to adjust the interactions between the variables to make the network more likely to generate the training data.

As a causal model we can say that for a belief net, on the contrary of B.M.'s, it is easy to get unbiased data, i.e., data which the model *believes* without training. By

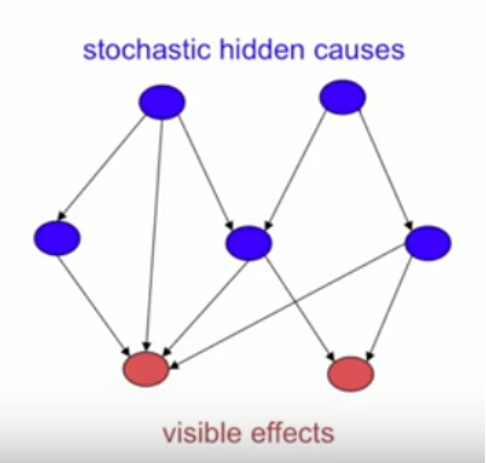

Figure 3: Example of Belief Net.

the way, the effect of explaining away makes harder to infer the posterior distribution over the hidden units, since they are, in some cases, strongly entangled by its previous causes. The searching for solving this problem leads up to a really important algorithm within the representation learning paradigm.

2.2.3 Greedy Layer-Wise Learning

The *greedy layer-wise learning* algorithm is proposed by Hinton et al. [2006], and has a crucial impact through the *deep learning* field evolution, "enabling researchers for the first time to train a deep supervised network without requiring architectural specializations like convolution or recurrence" [Goodfellow et al., 2016]. This algorithm can be characterized as a hibrid model, that relates both of the past two that we have just seen. This model uses a special case of B.M. on a certain way to find a good initialization for a joint successful learning procedure trough a deep belief net. We can characterize this idea as an *unsupervised pre training* canonical approach, such as a good point of start for the R.L. field study.

The model has "two top hidden layers that form a *undirected* assoative memory and the remaining hidden layers form an directed acyclic graph that converts the representations in the associative memory into observable variables" [Hinton et al., 2006], see section 3 for better understanding of these concepts. Between the atractives of the model are the easyness to interpret distributed representations, i.e., explaining away effects in the deep layers and also the fact that the learning algorithm is local.

To deal with the explaining away effect [Hinton et al., 2006] uses the elegant approach of a complementary prior for *sigmoid belief nets*, which are common belief nets with a stochastic property defined as

$$p(\mathbf{s}_i = 1) = \frac{1}{1 + \exp(-b_i - \sum_j \mathbf{s}_j w_{ij})}. \tag{23}$$

The argue continue as assuming that a belief net composed by just a unique hidden layer is factorial, because every *cause* comes directly from a unique unit, so the non-independence in the posterior distribution, i.e., the explaining away effects, is created by the likelihood term coming from the data. Then, using extra hidden layers to create a *complementary prior* that has exactly the opposite correlations to those in the likelihood term would eliminate the explaining away. So, multiplicating the likelihood to the prior would lead to the so desirable factorial posterior.

The most elegant part of the work made by Hinton et al. [2006] is demonstrating the fact that an infinite directed net can be equivalent to a Restricted Boltzmann Machine, which is a B.M. with just two layers and no relations between the units at the same layer. This overwhelming discovery resulted in a possible way to train deep belief nets, and also a manner to learn layers of features, which resulted in a big ascension on the *deep learning* field of study.

2.3 Depth

Alright, so the search for optimizing standard machine learning algorithms gave raise to a whole new branch of ideas, which many of them can be resumed as a several composition of functions connecting input layer and hidden layers. An interesting question is: Where are the representations we are talking about since we have started this text? And here it is what we call *semantics*, that is a quite complex theme which is not in focus here.

A good idea here is stated by Bengio and Delalleau [2011] "Deep Learning, i.e., learning multiple levels of representation. The intent is to discover more abstract features in the higher levels of the representation, which hopefully make it easier to separate from each other the various explanatory factors extent in the data", but this separation helps

mostly on supervision tasks. Two good advantages inside this kind of architectures are "deep architectures promote the re-use of features, and deep architectures can potentially lead to progressively more *abstract* features at higher layers of representations" [Bengio et al. [2013]]. Another interesting fact is that "More abstract concepts are generally invariant to most local changes of the input. That makes the representations that capture these concepts generally highly non-linear functions of the raw input" [Bengio et al. [2013]].

It is important to highlight the two main branches of models inside deep architectures. The first one are the ***probabilistic models***, and the second one is rooted in neural networks. The main difference between these models is about how we treat the layered architecture we mentioned above. While the probabilistic models use, in many cases, a graphical probabilistic approach, the neural nets treat the layered nodes as computational graphs. This can be resumed as the following question: "are hidden units considered latent variables or as computational nodes?" [Bengio et al., 2013]. Besides they are different approaches, it is interesting how they can be connected in some way, for example when we treat our hidden units as latent variables, i.e., a probabilistic model, the exact inference is typically intractable and, in some cases, unrolling *variational inference* to solve this results in a recurrent graph structure.

3 Probabilistic Approach

3.1 What is a Bayesian Approach?

Consider a typical case of trying to obtain a function $f(x)$, the classic probabilistic approach can be obtained starting from the statement $y(x) = f(x, \vec{\theta}) + \epsilon(x)$, where $\vec{\theta}$ is our vector of parameters and $\epsilon(x)$ is a *noise function*. The whole idea here is to construct a likelihood and looking for a manner to maximize it. Suppose that the noise function we choose is a standard normal $\mathcal{N}(0, \sigma^2)$. In this case we have the following conditional probability model for observation:

$$p(y(x)|x, \vec{\theta}, \sigma^2) = \mathcal{N}(y(x); f(x, \vec{\theta}), \sigma^2). \tag{24}$$

And the likelihood, for its turn, will be factorized as:

$$p(\vec{y}|x, \vec{\theta}, \sigma^2) = \prod_{i=1}^{n} \mathcal{N}(y(x_i); f(x_i, \vec{\theta}), \sigma^2). \tag{25}$$

Maximizing the likelihood with respect to σ^2 and $\vec{\theta}$ will lead us to:

$$\log p(\vec{y}|X, \theta, \sigma^2) \propto -\frac{1}{2\sigma^2} \sum_{i=1}^{n} [f(x_i, \theta) - y(x_i)]^2. \tag{26}$$

This expression looks like the *squared error function* on the equation (1), and it is a kind generalization of it, based on our prior choice for the noise function. This approach gives us a deterministic interpretation of the error measure and also gives us a sense of noise level σ^2. Since these methods are susceptible to over-fitting, which stands for a big accuracy on the training set and a high error on the test set, a manner to avoid it is using what we call *regularization*, by adding some kind of *complexity penalty* as we want, but how do we characterize this penalty? Looking for a way to obtain some interpretability on it we can think about maximizing the posterior using a gaussian prior $p(\vec{\theta})$ we have

$$\log p(\vec{\theta}|\vec{y}, X) = \log p(\vec{y}|\vec{\theta}, X) + \log p(\vec{\theta}) + c. \tag{27}$$

So the log prior could be interpreted as a regularizer. But this is not *Bayesian*!

On the other side, a *Bayesian* proposal is regarded to attain a predictive distribution which is not dependent on the parameters $\vec{\theta}$. Using the *sum rule* and the *product rule* for probabilities, a way to obtain this expression we are looking for is using a *marginalization* over the parameters, so the distribution takes the following form:

$$p(y|x_*, \vec{y}, X) = \int p(y|x_*, \vec{\theta}) p(\vec{\theta}|\vec{y}, X) d\vec{\theta}$$

This kind of proposal has attributes of generalization. If we think on each setting of θ as a model, this approach gives us a *Bayesian model average* of infinitely many models weighted by their posterior probabilities, this represents what we call *epistemic uncertainty* over which the function $f(x, \theta)$ fits the data. This approach gives us robustness against over-fitting, since there are many different functions corresponding to different

settings of the parameters and we are not sure given a finite sample which is the right description of the data. Besides that, if we set an approximate as $q(\vec{\theta}|\vec{y}, X) = \delta(\theta = \theta_{MAP})$ we generalize classical training either.

3.2 Graphical Models

Instead of searching for mapping functions or automating classification tasks, the approaches we are about to expose here are concerned to obtain *distributions* from what we called on the past section as *hidden units*, by the way it is important to notice that the distributions we are looking for are always related to a given task. So, we interpret these hidden units as *latent random variables* $\mathbf{h}$, using the observed data $\mathbf{x}$. "We can express as $p(x, h)$ a probabilistic model over the joint space of the latent variables, $\mathbf{h}$, and observed data $\mathbf{x}$ or visible variables. Feature values are conceived as the result of an inference process to determine the probability distribution of the latent variables given the data, i.e., $p(h|x)$, often referred to as the ***posterior probability***" [Bengio et al., 2013].

One of the possible ways to reach these ideas is using *graphical models*, which "provide a formal framework for modeling only direct interactions between random variables. This allows the models to have significantly fewer parameters and therefore be estimated reliably from less data." Goodfellow et al. [2016]. The usage of only direct interactions between our hidden units has several motivations, as for example requiring a smaller computational cost and also using the hypothesis that, in many cases, there is no need for using every interaction, since the influence of hidden units which are not directly connected can be inherited through the path that connects them. A good example to clarify here, exposed in Goodfellow et al. [2016], is the problem of modeling finishing times of a team with three people in a relay race, for these kind of problems, for example, it is clear that the influence of time expended between the first runner on the third one is inherited though the second one, so modeling using just directly interactions would be really valid here.

Graphical models, or G.M., are one of the manners for describing probabilistic distributions, using graph structures. An important highlight here is that the assumptions about how the random variables interact with each other can be represented in these type

of models, an this is one of the reasons that makes them good choices in some cases. For this work a *graph* $G = (\mathcal{V}, \mathcal{E})$ consists of a set of *nodes*, $\mathcal{V} = \{1, ..., V\}$, and a set of *edges*, $\mathcal{E} = \{(s,t) \in \mathcal{V}\}$. We write $G(s,t) = 1$ to denote $(s,t) \in \mathcal{E}$, that is, if $s \to t$ is an edge in the graph. We also use the notation for *parents* of a generic node s as $pa_G(s) \triangleq \{t : G(t,s) = 1\}$, and *children* of a generic node s as $ch_G(s) \triangleq \{G(s,t) = 1\}$. Another definition we need here is *neighbors* of a node, which stands for the set of all immediately connected node, $nbr(s) \triangleq \{t : G(s,t) = 1 \wedge G(t,s) = 1\}$. So the idea is that, with graphical models, it is possible to represent random variables as nodes and the the interactions between them as edges, where each edge represents a direct interaction of two random variables. There are two main branches of graphical models in this context that we will expose here, *direct graphical models* and *indirect graphical models*.

3.2.1 Directed Graphical Models

Directed graphical models are characterized as models that are constructed over a *directed acyclic graph*, or DAG, structure. Saying that a graph is acyclic means that this graph has no *cycles*, i.e., a series of nodes such that we can get back to where we started by following edges. From the other side, a directed graph is a graph whose edges are directed, i.e., they point from one node to another. An important attribute of these models is that *conditional independence* assumptions can be represented within his structure, such as other interesting assumptions. For example, the directed edges can represent *causal relations*. A property of DAG's is that nodes can be ordered such that parents come before children, i.e., a *topological ordering* can be constructed in any DAG.

Conditional independence assumptions are strongly related with the factorization of conditional marginal distributions, i.e., $X \perp Y|Z \iff p(X,Y|Z) = p(X|Z)p(Y|Z)$. Then, the manner we construct our graph allows us to define ***local conditional probability distributions*** $p(h_i|pa_G(h_i))$, so the probability distribution over $\mathbf{h}$ is given by

$$p(\mathbf{h}) = \prod_i p(h_i|pa_G(h_i))$$

The idea behind this formalism in the context of the R.L. field is, in many cases, to separately parametrize the conditional likelihood $p(x|h)$ and the prior $p(h)$ to construct

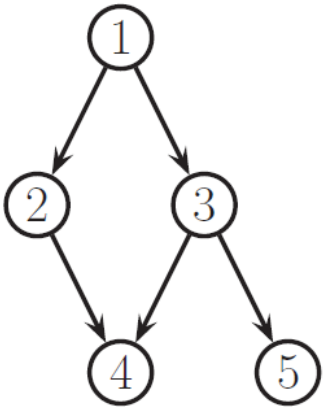

Figure 4: Example of Directed Acyclic Graph, extracted from Murphy [2013]

the joint distribution $p(x, h) = p(x|h)p(h)$. An important consequence of these approaches is that the property of representing the indirect influences of random variables using only parent nodes has a strong attribute of *explaining away*, i.e., "a priori independent causes of an event can become non-independent given the observation of the event. Latent factor models can be generally be interpreted as latent *cause* models, where the h activations cause the observed x" [Bengio et al., 2013], and we know that distributed representations are good for our purpose. Although, while this expression of dependence looks great, life is not so easy because the recover of the posterior $p(h|x)$ can potentially become intractable. Furthermore, we need to consider that it could exist several assumptions which it wouldn't be capable to be expressed by the way we dispose our graph, in a simple way we can say that the graph structure, in this case, is regarded only to conditional independence assumptions.

3.2.2 Undirected Graphical Models

The *undirected models* is another approach for describing probabilistic models. But in this case we will consider that the edges are not directed, this simple assumption changes everything and gives support to many interesting interpretations. An undirected graphical model stands for a probabilistic model defined on a undirected graph.

In this case it is possible to make assumptions related to conditional independence using the manner that the variables, or set of variables, separates each

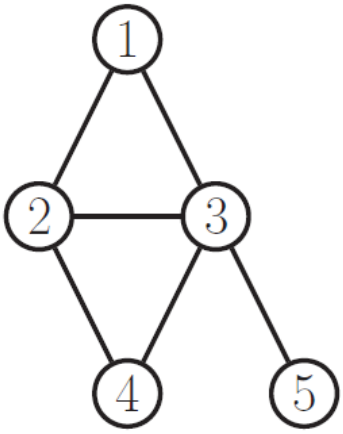

Figure 5: Example of Undirected Graph, extracted from Murphy [2013]

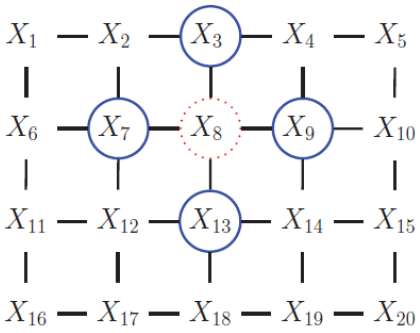

Figure 6: Example of Undirected Graph, in this case the red node X_8 is independent of the other black nodes given its neighbors (blue nodes). Extracted from Murphy [2013]

.

other in the graph. For example, for sets of nodes A, B and C in a graph G, we say $\mathbf{h}_{AG}\mathbf{h}_B|\mathbf{h}_C \iff$ C separates A from B in the graph G With these kind of ideas it is possible to construct statements called *Markov properties*, which by them we derive the conditional independence properties of the graph. A manner to divide our undirected graph is dividing it by *cliques*. For an undirected graph a clique is a set of nodes that are all *neighbors* of each other, and this is our starting point. For each *clique* $\mathcal{C}$ in the graph, a non-negative *clique potential* $\phi(\mathcal{C})$ is regarded to how the random variables within that clique relates with each other in a joint state. The set of cliques within these graphs

defines the *unnormalized probability distribution*

$$\tilde{p}(\mathbf{h}) = \prod_{\mathcal{C} \in G} \phi(\mathcal{C}). \tag{28}$$

Here is important to notice that, until now, the clique potentials does not necessarily carry a probability distribution meaning. We need to normalize the potentials setting a *partition function* Z defined, in a continuous case, by

$$Z = \int \tilde{p}(\mathbf{h}) d\mathbf{h}. \tag{29}$$

Notice that this integral depends directly on our choice for the potentials and also it is over all the possible assignments for $\mathbf{h}$. This fact can potentially lead us to a intractable partition function.

By making a good choice for the potentials and random variables, or even dealing with intractability, the idea of these approaches within the R.L. field is that, in many cases, it is possible to parametrize the joint distribution $p(x, h)$ through a product of non-negative *clique potentials*.

$$p(x, h) = \frac{1}{Z_\theta} \prod_i \phi_i(x) \prod_j \eta_j(h) \prod_k \nu_k(x, k), \tag{30}$$

where $\phi_i(x)$, $\eta_j(h)$ and $\nu_k(x, k)$ are the clique potentials describing the interactions between the visible elements, between the hidden variables, and those interaction between the visible and hidden variables respectively. Perceive that the partition function is defined in terms of the parameters θ. The particular choice we wish to expose here in this work is the following one

$$p(x, h) = \frac{1}{Z_\theta} \exp -E_\theta(x, h). \tag{31}$$

The models with this proposal are known as *energy based models*, where $E_\theta(x, h)$ is called the energy function, which stands for the manner that the clique potentials describe the interactions between the random variables, using the parameters θ. Notice that this choice makes that the product of clique potentials potentially become sums, and this is an interesting property. But be careful, the tractability of the partition

function still depends on the choice for potentials so as the random variables that we are dealing with, for example the difference between dealing with a continuous or discrete random variables would lead us to totally different models.

3.3 Dealing with Intractable Posteriors

In many cases, feature extraction tasks using latent variable models can be constructed by maximizing likelihood assumptions, which is characterized as *inference* approaches. As mentioned before, the construction of either directed and undirected graphical approaches are, in many cases, subjected to lead us to intractable distributions. The eventually difficult for dealing with these kind of proposal motivates us to expose here an alternative for dealing with intractable distributions, variational inference based on Murphy [2013].

3.3.1 Variational Inference and the Mean Field Method

The basic idea behind variational inference, for dealing with an intractable distribution, is to pick an approximation q($\mathbf{x}$) to the distribution from some tractable family, and then try to make this approximation as close as possible to the true posterior given data, $p^*(\mathbf{x}|\vec{D})$. This approach relates inference and optimization, and allows us to trade accuracy for speed by relaxing constraints. Suppose that $p^*(\mathbf{x})$ is our true but intractable distribution and $q(\mathbf{x})$ the tractable approximation with some free parameters. A good idea is to minimize the KL divergence

$$\mathbb{KL}(p^*||q) = \sum_{\mathbf{x}} p^*(\mathbf{x}) \log \frac{p^*(\mathbf{x})}{q(\mathbf{x})}, \tag{32}$$

because we know that p^* is intractable it sounds better to use the reverse KL divergence, so as we could compute expectations with respect to q

$$\mathbb{KL}(q||p^*) = \sum_{\mathbf{x}} q(\mathbf{x}) \log \frac{q(\mathbf{x})}{p^*(\mathbf{x})}. \tag{33}$$

The approximation of the true posterior $p^*(\mathbf{x}) = p(\mathbf{x}|\vec{D})$ depends on the normalization $Z = p(\vec{D})$. However, an interesting idea is that, usually, the unnormalized distribution

$\tilde{p}(\mathbf{x}) \triangleq p(\mathbf{x}, \vec{D}) = p^*(\mathbf{x})Z$ is tractable to compute. So we set our new objective function as

$$J(q) \triangleq \mathbb{KL}(q||\tilde{p}) \tag{34}$$

$$J(q) = \sum_{\mathbf{x}} q(\mathbf{x}) \log \frac{q(\mathbf{x})}{\tilde{p}(\mathbf{x})} \tag{35}$$

$$J(q) = \sum_{\mathbf{x}} q(\mathbf{x}) \log \frac{q(\mathbf{x})}{Zp^*(\mathbf{x})} \tag{36}$$

$$J(q) = \sum_{\mathbf{x}} q(\mathbf{x}) \log \frac{q(\mathbf{x})}{p^*(\mathbf{x})} - \log Z \tag{37}$$

$$J(q) = \mathbb{KL}(q||p^*) - \log Z. \tag{38}$$

So the minimizing of $J(q)$ will lead us to approximating q to p^*. Since the KL divergence is always non-negative, we see that J(q) is an upper bound on the negative log likelihood:

$$J(q) = \mathbb{KL}(q||\hat{p*}) - \log Z \geq -\log Z = \log p(\vec{D}), \tag{39}$$

notice that variational inference is closely related to expectation maximization.

Another interesting idea is to assume that the posterior is a fully factorized approximation of the following form

$$q(\mathbf{x}) = \prod_i q_i(\mathbf{x}_i), \tag{40}$$

so the minimization turns to

$$\min \mathbb{KL}(q_i||p) \tag{41}$$

over the parameters of each marginal distribution q_i. At each step we make the following update:

$$\log q_j(\mathbf{x}_j) = \mathbb{E}_{-q_j}[\log \tilde{p}(\mathbf{x})] = \text{const}, \tag{42}$$

where $\tilde{p}(x) = p(\mathbf{x}|\vec{D})$ is unnormalized posterior and the notation $\mathbb{E}_{-q_j}[f(\mathbf{x})]$ means to take the expectation over $f(\mathbf{x})$ with respect to all variables except for x_j.

4 Conclusions and Motivations

Considering the purposes of our work, the main conclusions are regarded to how the exposed content behaves as one entire piece and how it relates with the reader and writer. By the method that we present the probabilistic models it is possible to construct a clear timeline, restricted to some of the big events of the machine learning field as a whole, and also to the themes that are, in many cases, strongly connected with the representation learning area. It is clear that the two main generative models exposed in section 2 converge to the work made by Hinton et al. [2006], and this is the most impressive fact exposed in our work: How these approaches, which are related by one side to physics and by the other side to statistics, helped to lead the deep learning paradigm in which we live today.

The models presented in section 3 have a strong relation with the broken paradigms exposed in section 2. Also, with the tone that the math is constructed, it is possible to use this work to initiate a research in some real or theoretical problem, and this is one of the motivations that we had. Besides that, we have the wish to continue the study on the R.L. theme, but with a more focused approach on one of the main areas on machine learning, such as NLP or Computer Vision. The main idea is to use this work, like the title suggests, as a *starting point*.

References

Y. Bengio and Olivier Delalleau. On the expressive power of deep architectures. page 1, 10 2011. doi: 10.1007/978-3-642-24412-4_3.

Yoshua Bengio, Aaron C. Courville, and P. Vincent. Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35: 1798–1828, 2013.

Terrence J. Sejnowski David H. Ackley, Geoffrey E. Hinton. A learning algorithm for boltzmann machines. *Cognitive Science*, 9:147–169, 1985.

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

Geoffrey E. Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural Comput.*, 18(7):1527–1554, July 2006. ISSN 0899-7667. doi: 10.1162/neco.2006.18.7.1527. URL `https://doi.org/10.1162/neco.2006.18.7.1527`.

Kevin P. Murphy. *Machine learning : a probabilistic perspective*. MIT Press, Cambridge, Mass. [u.a.], 2013. ISBN 9780262018029 0262018020.

Radford M. Neal. Connectionist learning of belief networks. *Artificial Intelligence*, 56(1): 71–113, 1992. doi: 10.1016/0004-3702(92)90065-6.