Deep Learning: A Statistical Perspective

Pietro Lesci

Latest version: November 2018

Abstract

Blank

Blank

Acknowledgment

Blank

Declaration

Blank

Contents

List of Figures

Notation

The next list describes several symbols that will be later used within the body of the document

Numbers and Arrays

x	A scalar or a scalar random variable
D	Input dimesionality
$\boldsymbol{x}$	A deterministic vector
$\mathbf{x}$	A random vector
$\mathbf{A}$	A matrix
$\boldsymbol{W}$	The weight matrix
$\boldsymbol{I}$	Identity matrix
$\mathbf{X}$	Dataset inputs (matrix with N rows, one for each data point)
L	Number of network layers
N_l	Number of network units in layer l

Set and Graphs

$\mathbb{R}$	The set of real numbers
$\{0, 1, \ldots, n\}$	The set of all integers between 0 and n

Indexing

a_i	Element i of vector $\boldsymbol{a}$, or random vector $\mathbf{a}$; indexing starting at 1
A_{ij}	Element i, j of matrix A
$A_{i,:}$	Row i of matrix A
$A_{:,i}$	Column i of matrix A

Chapter 1

Introduction
Deep Learning: The Algorithmic Perspective

"Recall seeing a package to make quotes"

Snowball

The desire to create machines capable of thinking accompanies the human kind since the first programmable computer was invented. Nowadays, artificial intelligence (AI) is a thriving field of research that encompasses various disciplines such as computer science, engineering, statistics, neuroscience, biology, with applications that ranges from automating routine labour, interpreting images, understanding speech, to making diagnoses in medicine.

Problems that are intellectually difficult for humans to tackle, namely those problems that can be described by a list of formal, mathematical rules, are among the easiest for computers to solve. The real challenge to AI is the resolution of tasks relatively easy for people to perform, but hard to describe in formal terms - problems that we, as human beings, solve intuitively, instinctively - such as recognising objects in images. This instinctive "intuitions" are drawn from experience: collections of small facts (data) and personal judgements of facts (information).

One solution to let machines mimic the process of deduction from experience is to hard-code the knowledge about the reality in formal language, allowing the computer to reason using logical inference rules. This approach, called **knowledge based** [4], is brute-force in nature and not viable in many practical applications.

Therefore, AI systems need to have the ability to "build" their own knowledge by extracting information from raw data. This field of research is known as **machine learning**. However, the performance of these algorithms crucially depends on the *representation* of data they are fed with - the usual maxim "garbage in, garbage out". Each bit of information included in the representation is called **feature**[1]. Selecting or

[1]In the econometric literature *features* are known as *explanatory variables* or *regressors*.

extracting features, that forms representations, from raw data is a form of art of its own. Often, as alternative to hand-designed features, machine learning algorithms are used to learn such *representations* besides learning the *mapping* from representations to outputs. This approach is called **representation leaning** [4]. Regardless the methodology used to build such features, the aim is often to separate the **factors of variation** that explain the observed data. In many cases, these factors are not directly observed: "they may exist as either unobserved objects or unobserved forces [...] constructs in the human mind [...] concepts or abstractions that help us make sense of the rich variability in the data"[2]. Therefore, extracting representations can be as difficult as solving the original problem. In these cases representation learning comes unhandy.

Deep Learning provides a solution by taking a constructionist approach: creating complex, expressive representations of the world in terms of simple, more basic ones. More formally, deep learning can be defined as "[...] a form of machine learning that uses hierarchical abstract layers of latent variables to perform pattern matching and prediction"[3]. In the next sections we explain why and how this approach works.

In this chapter we provide the definition of learning algorithm (§1.1) and a collection of related concepts fundamental to understand the matter.

1.1 Learning algorithm

An informative, albeit succinct, definition of learning algorithm is provided in [10]: "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E". Of course, the variety of experiences, tasks, and performance measures is wide. Below a high-level overview of the possible instances of this concepts is provided:

- Tasks T: classification, classification with missing inputs, regression, anomaly detection, imputation of missing values, denoising, density estimation
- Performance measures P: accuracy, error rate, loss functions
- Experiences E: supervised, unsupervised, semi-supervised, and reinforcement learning

As a concrete example, consider the linear regression; in doing this we introduce also the jargon of the machine learners trying to "map" it to the pure statisticians' one. The usual environment a machine learner or a statistician lives into in practise is the following: the data are gathered into a **dataset** - that can be hosted on a single computer, in a database, or in a collection of databases known as cluster (of databases) - which is a collection of many **examples**, or observations. An example is, in turn, a collection of features, or possible explanatory variables, related to an object or event that have been quantitatively measured called **label**, target, or dependent

[2] [4] pagg. 4-5.
[3] [11] pag. 1.

variable. Often the dataset is split into **training** and **test** sets: the former contains examples that are used to train the algorithm, the latter contains examples used to test its "out-of-sample" performances. Usually, there is a third partition called the **validation** set, used to learn the hyperparameters of the model.

In case of linear regression, as the name suggests, the task T is regression, i.e. predict a numerical value given some inputs; the performance measure P could be the mean squared error; and the experience E is represented by the examples of the training set. Each example is associated with a label that lets the algorithm compare its prediction with the observed datum (the "truth"). Algorithms that are trained via comparisons of prediction vis-à-vis actual observation go under the name of **supervised learning** algorithms - linear regression belongs, thus, to this class. We are, thus, presented with a collection of labelled examples $\{\boldsymbol{x}_i, y_i\}_{i=1}^N$, where N is the size of the collection, $\boldsymbol{x_i} \in \mathbb{R}^D$ is the D-dimensional feature vector of the example, or observation, $i = 1, \ldots, N$, $y_i \in \mathbb{R}$ is a real valued target and every feature x_j, $j = 1, \ldots, D$, is also a real number.

We want to build a model $f_{\boldsymbol{w}}(\boldsymbol{x_i})$ as a linear combination of features that best matches the labels

$$f_{\boldsymbol{w}}(\boldsymbol{x_i}) = \boldsymbol{w}\boldsymbol{x_i}$$

where $\boldsymbol{w}$ is a D-dimensional vector of parameters, known as **weights**, and $\boldsymbol{w}\boldsymbol{x_i}$ is a *dot-product*. The notation $f_{\boldsymbol{w}}$, means that the model f is parametrised by the value $\boldsymbol{w}$. Usually, a modified version of the model is used: an intercept term $b \in \mathbb{R}^D$ is added in the equation

$$f_{\boldsymbol{w},b}(\mathbf{x_i}) = \boldsymbol{w}\boldsymbol{x_i} + b$$

This still preserves the linear mapping from parameters to predictions, but the mapping from feature to predictions is now an affine function[4]. The intercept, in the machine learning jargon, is often called the **bias**[5] [4] parameter (of the affine transformation) or **offset vector** [11]. This nomenclature derives from the fact that, in the absence of any input, the transformation is biased towards b.

The model is used to predict the unknown $y^\star$ for a given $\boldsymbol{x}^\star$ like this: $y^\star = f_{\boldsymbol{w},b}(\boldsymbol{x}^\star)$. Two models parametrised by two different pairs $(\boldsymbol{w}', b')$ will likely produce two different predictions when applied to the same example. Therefore, we want to find the optimal values $(w^\star, b^\star)$. This procedure is called **optimisation**. Usually parameters are optimised on the training set and the out-of-sample predictions are tested on the test set.

The linear regression example lets us define the "anatomy" of a learning algorithm:

1. A loss function (e.g. mean squared errors)
2. An optimisation criterion based on the loss function (e.g. maximum likelihood)
3. An optimisation routine that leverages training data to find a solution to the optimisation criterion (e.g. gradient descent)

[4]Albeit trivial as a remark, it will be useful in what follows, especially when defining deep learning formally.

[5]This term is not related whatsoever to the idea of *statistical bias*.

Reading modern literature [5, 1] on machine learning, we often encounter references to **gradient descent** or **stochastic gradient descent**. These are two most frequently used optimisation algorithms when the optimisation criterion is differentiable. Gradient descent is an iterative optimisation algorithm for finding the minimum of a function.

In general, a learning algorithm - also referred to as learning machine - is an input-output mapping "learned" from data itself

$$\boldsymbol{y} = f(X)$$

where $X = [\boldsymbol{x}_1 \cdots \boldsymbol{x}_K]^\top \in \mathcal{M}(N, K)$, the space of $M \times K$ matrices, often called **design matrix**, containing a different example in each row. Linear regression is a particular case of learning algorithm that, by construction, can only capture linear mappings, or functions, between inputs and outputs. Informally, in the machine learning lingo, the ability to fit certain variety of functions is called **capacity**- e.g. linear regression as a lower capacity than nonlinear regression. Furthermore, the set of functions that the learning algorithms able to approximate is referred to as **hypothesis space** - e.g. the hypothesis space of linear regression are all linear functions of its inputs in case the intercept is present, otherwise it is the set of all linear function passing through the origin of the Cartesian axes.

As for any statistical and econometric procedure, the main goal of a learning algorithm is to perform well, in term of the performance measure used, on new data, that is reach good performances on the training set. Besides the minimising the **training error** while "learning" - often identified by the error in prediction on the training set - a learning algorithm aims at generalising well, that is minimising the **generalisation error** - usually identified by the error rate on the test set. Therefore, the objective of a learning algorithm is twofold, that is minimising (according to a performance measure):

- **Underfitting**: high error rate on the training
- **Overfitting**: large gap between training error and generalisation error

The hype for deep learning algorithms is due their great performances, with respect to their fellow machine learning algorithms, in tasks like speech recognition and object detection for AI purposes. Such endeavours are characterised by high-dimensional data to digest and complicated functions in high-dimensional spaces to approximate. This entails high computational costs and issues in generalising to new examples. Deep learning provides a way to overcome such difficulties.

In the next section, we provide a more formal introduction and we provide, as examples, some instances of the class of algorithms that goes under the name of deep learning.

1.2 Deep learning: the algorithm

The term "Deep Learning" (DL, henceforth) was first introduced in the machine learning (ML henceforth) in 1986, and later used for Artificial Neural Networks (ANN,

henceforth) in 2000 [12]. In general, a deep learner - as opposed to shallow learner - is a representation learning methods with multiple levels of representation, obtained by composing simple but nonlinear transformations of the raw data, with each of these transforming the representation at one level (starting with the raw input) into a representation at a higher level of abstraction. Composing enough of such transformations, very complex functions can be learned [8]. A DL architecture can be thought as a multilayer stack of simple models - generalised linear models [11] - subject to learning, that is, parameters need to be estimated.

Similarly to any other learning algorithm, a deep learner is an input-output mapping, $f : \mathcal{X} \to \mathcal{Y}$, where the input component, $\boldsymbol{X} \in \mathcal{X}$, is high-dimensional. The output component, $\boldsymbol{y} \in \mathcal{Y}$, can be discrete (e.g. classification), continuous (e.g. regression) or mixed. The goal is to approximate

$$\boldsymbol{y} = f(\boldsymbol{X})$$

by applying, in each layer, $l = 1, \ldots, L$, the composition of an affine transformation[6] of the inputs, controlled by learned parameters $(\boldsymbol{W}, \boldsymbol{b})$, and a fixed nonlinear function applied element-wise, known as **activation function**. The output of any intermediate layer is the input of the successive one. The chain ends when the output layer is reached. Every intermediate layer, namely those layers that are neither the input or output layer, are called **hidden layers**, the dimesionality D_l of the hidden layers defines the **width** of the model, while the overall number of hidden layers determines the **depth** of the model. The name "deep" learning has arisen from this kind of terminology. The class of models described above as composition of functions is referred to as **neural networks**. It is a two-stage regression or classification model: in the first stage the features are derived via a linear combination of the inputs; in the second stage the target output is modelled as a nonlinear function of these extracted features [5]. These models are associated with **directed acyclic graphs**, also known as *network diagram* in the machine learning literature, describing how functions are composed together. Chain structures, as the one described above, are the most common structures in deep learning [4]. See figure 1.1 for a typical directed acyclic graph (network graph) representing a neural network structure; hidden layers are highlighted in gray.

The formal roots of DL can be traced back in Kolmogorov's representation of a multivariate response surface as a superposition of univariate activation functions applied to an affine transformation of the input variables [7]. In 1957 the Russian mathematician Kolmogorov showed that *any* continuous function f of many variables can be represented as a composition of addition and some functions of one variable. The original version of this theorem can be expressed as follows[7]:

Theorem 1 (Kolmogorov–Arnold superposition theorem) *Let $f : [0,1]^d \to \mathbb{R}$ be an arbitrary multivariate continuous function defined on the identity hypercube.*

[6] An affine transformation of a vector is a weighted sum of its elements (linear transformation) plus an offset constant (bias).

[7] We propose a version of the theorem deducted from [2, 9].

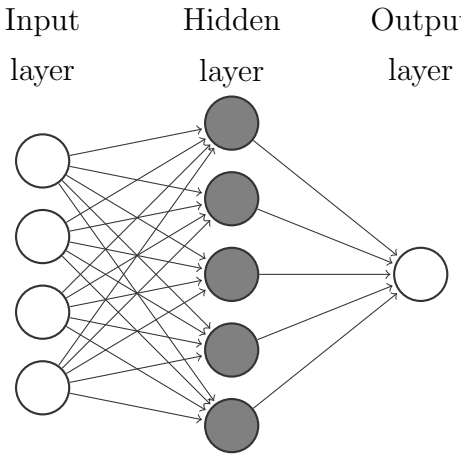

Figure 1.1: *Stylised neural network diagram composed by one hidden layer, highlighted in gray. The input layer has four neurons, while the hidden layer has five neurons. They correspond to the components of the input vector and of the transformed input vector, respectively. Each circle is a neuron which calculates a weighted sum of an input vector plus bias and applies a nonlinear function to produce an output.*

Then it has the representation

$$f(\boldsymbol{x}) = f(x_1, \ldots, x_d) = \sum_{q=0}^{2d} \phi_q \left(\sum_{p=1}^{d} \psi_{q,p}(x_p) \right)$$

with continuous one–dimensional inner and outer functions ϕ_q and $\psi_{q,p}$, defined on the real line. The inner functions $\psi_{q,p}$ are independent of the function f.

Recently [9], Kolmogorov's superposition theorem found attention in neural network computation by Hecht–Nielsen's interpretation as a feed-forward network with an input layer, one hidden layer and an output layer [6]. However, the inner functions in all these versions of the theorem are highly nonsmooth. Sprecher [13] explicitly describes construction methods for univariate functions, introduces interesting notions for theorem comprehension. Bryant [3] implements Sprecher's [14] algorithm to estimate the nonsmooth inner link function. Intuitively, deep layers allow for smooth activation functions to provide "learned" hyperplanes which find the underlying complex interactions and regions without having to see an exponentially large number of training samples.

Given a high-dimensional input[8] $\boldsymbol{X} = [\boldsymbol{x}_1 \cdots \boldsymbol{x}_D] \in \mathbb{R}^{N \times D}$ a deep learner finds a predictor $\hat{\boldsymbol{y}} \in \mathbb{R}^N$ of an output $\boldsymbol{y} \in \mathbb{R}^N$. Therefore, it computes an input-output mapping, $\boldsymbol{y} = f(\boldsymbol{X})$, where the input space is high-dimensional. The high-dimensional mapping, $f(\boldsymbol{X})$, is modelled via the superposition of univariate semi-affine functions where univariate activation functions to decompose the high-dimensional $\boldsymbol{X}$.

[8]More correctly, we should use the notation $\mathcal{M}(N, D)$ the space of $N \times D$ matrices. We are implicitly applying the isomorphism $iso : \mathcal{M}(N, D) \to \mathbb{R}^{N \times D}$ between two isomorphic vector spaces. We are, thus, vectorising the matrix X.

Let $\varphi_1, \ldots, \varphi_L$ be nonlinear univariate activation functions such that $\varphi_l : \mathbb{R} \to \mathbb{R}$. A semi-affine activation rule is given by [11]:

$$f_l^{\boldsymbol{W}_l, \boldsymbol{b}_l} = \varphi_l \left(\sum_{i=1}^{N_l} \boldsymbol{W}_{ij}^{(l)} h_j + \boldsymbol{b}^{(l)} \right), \quad j = 1, \ldots, N_l, \quad l \in \{1, \ldots, L\}$$

where $\boldsymbol{W}$ and $\boldsymbol{h}$ are the weight matrix and inputs of the lth layer, respectively, and N_l is the dimension of the input vector at each layer, namely the number of hidden units that are selected using a stochastic search technique known as **dropout**. The deep predictor can be written as

$$\hat{\boldsymbol{y}}(\boldsymbol{X}) \coloneqq \left(f_1^{\boldsymbol{W}_1, \boldsymbol{b}_1} \circ \cdots \circ f_L^{\boldsymbol{W}_L, \boldsymbol{b}_L} \right)(\boldsymbol{X})$$

Let $\boldsymbol{X} = \boldsymbol{h}^{(0)}$. Then a deep prediction rule can be expressed as:

$$\begin{aligned}
\boldsymbol{h}^{(1)} &= \varphi_1 \left(\boldsymbol{W}^{(0)} \boldsymbol{X} + \boldsymbol{b}_0 \right) \\
\boldsymbol{h}^{(2)} &= \varphi_2 \left(\boldsymbol{W}^{(1)} \boldsymbol{h}^{(1)} + \boldsymbol{b}_1 \right) \\
&\cdots \\
\boldsymbol{h}^{(L)} &= \varphi_L \left(\boldsymbol{W}^{(L-1)} \boldsymbol{h}^{(L-1)} + \boldsymbol{b}_{L-1} \right) \\
\hat{\boldsymbol{y}}(\boldsymbol{X}) &= \boldsymbol{W}^{(L)} \boldsymbol{h}^{(L)} + \boldsymbol{b}_L
\end{aligned}$$

To summarise, the purpose of deep learning is to extend linear models to represent nonlinear functions of the inputs, X, applying the linear model not to $\boldsymbol{X}$ directly, but to a transformed input, $\varphi(X)$, where φ is a nonlinear transformation. Deep learning, rather than manually engineering this transformation, learns it. It is important to note that the behaviour of the hidden layers is not directly specified in the data, it is learned. The learned mapping from X to y is **deterministic**.

How learning is performed

Bibliography

Bishop, Christopher M. (2006). *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag.

Braun, Jurgen and Michael Griebel (2009). "On a Constructive Proof of Kolmogorov's Superposition Theorem". In: *Constructive Approximation* 30.3, p. 653. URL: %5Curl%7Bhttps://doi.org/10.1007/s00365-009-9054-2%7D.

Bryant, Donald W. (2008). "Analysis of Kolmogorov's superposition theorem and its implementation in applications with low and high dimensional data". In:

Goodfellow, I., Y. Bengio, and A. Courville (2016). *Deep Learning*. http://www.deeplearningbook.org. MIT Press.

Hastie, Trevor, Robert Tibshirani, and Jerome Friedman (2009). *The Elements of Statistical Learning*. 2nd ed. Springer Series in Statistics. New York, NY, USA: Springer New York Inc.

Hecht-Nielsen, R. (1987). "Kolmogorov's mapping neural network existence theorem". In: *Proceedings of the International Conference on Neural Networks III*, pp. 11–14.

Kolmogorov, A. N. (1957). "n the representation of continuous functions of many variables by superpositions of continuous functions of one variable and addition". In: *Doklay Akademii Nauk USSR* 14.5, pp. 953–956.

LeCun, Y., Y. Bengio, and G. E. Hinton (2015). "Deep learning". In: *Nature* 521.7553, pp. 436–444.

Leni, P. E., Y. Fougerolle, and F. Truchetet (2011). "Kolmogorov Superposition Theorem and its application to multivariate function decompositions and image representation". In: ed. by USA IEEE Computer Society Washington DC.

Mitchell, Thomas M. (1997). *Machine Learning*. 1st ed. New York, NY, USA: McGraw-Hill, Inc.

Polson, N. and V. Sokolov (2017). "Deep Learning: A Bayesian Perspective". In: *ArXiv e-prints*. http://adsabs.harvard.edu/abs/2017arXiv170600473P.

Schmidhuber, Jurgen (2015). "Deep learning in neural networks: An overview". In: *Neural Networks* 61, pp. 85–117.

Sprecher, D. A. (1965). "On the structure of continuous functions of several variables". In: 115.3. Ed. by Amer. Math. Soc, pp. 340–355.

— (1972). "A survey of solved and unsolved problems on superpositions of functions". In: *Journal of Approximation Theory* 6.2, pp. 123–134.

Appendix A

KL condition