Mathematical Perspective of Machine Learning

Yarema Boryshchak, PhD

December 18, 2020

Abstract

We take a closer look at some theoretical challenges of Machine Learning as a function approximation, gradient descent as the default optimization algorithm, limitations of fixed length and width networks and a different approach to RNNs from a mathematical perspective.

1 Introduction

In the nutshell the idea of training a neural network (**NN**) is equivalent to the problem approximation of a given function, $\mathfrak{f}$, with the domain, $\boldsymbol{\mathcal{D}}$, and codomain, $\boldsymbol{\mathcal{C}}$,

$$\mathfrak{f} : \mathcal{D} \to \mathcal{C} \tag{1}$$

which depends on some data of size $N \in \mathbb{N}$ of k-dimensional input vectors, $\boldsymbol{\vec{x}_j} \in \mathcal{D} \subseteq \mathbb{R}^k$ and l-dimensional output (label) vectors, $\boldsymbol{\vec{y}_j} \in \mathcal{C} \subseteq \mathbb{R}^l$, by a composition of functions of the form

$$\vec{P}_i\left(\vec{z}_{i-1} \cdot \vec{w}_i\right) = \left(P_i\left(\vec{z}_{i-1} \cdot \vec{w}_i\right), \dots, P_i\left(\vec{z}_{i-1} \cdot \vec{w}_i\right)\right), \tag{2}$$

where $\boldsymbol{P_i}$ is called an activation function of layer i, $\vec{z}_{i-1}$ is an output vector of the layer $i-1$, and $\boldsymbol{\vec{w}_i}$ is called the weight vector of layer i. Once the size of each layer and the choice of each activation function is made, one usually uses, so called, back propagation algorithm, adjusting the values of each weight vector according to some type of gradient descent rule. In other words, one is trying to solve an optimization problem, minimizing the "difference norm"

$$\mathfrak{C} := \left\|f - \tilde{f}\right\|_d, \qquad \text{where} \qquad \boldsymbol{\tilde{f}} = \vec{P}_r\left(\vec{P}_{r-1}(\dots \vec{P}_1)\right) \tag{3}$$

and $r \in \mathbb{N}$ is the number of layers of the neural network.

So apriori, we are making a choice of the function $\tilde{f}$ of weights $\vec{w}_1, \dots, \vec{w}_r$. Note that the dimension of each vector $\vec{w}_i$ is the size of the layer i. To simplify the problem we may always find the maximum, m, of the size layers, and assume that each $\vec{w}_i \in \mathbb{R}^m$. We are implicitly assuming that for each $i = 1, ..., r$, $\vec{P}_i(\vec{0}) = \vec{0}$. [1]

2 Existence of function $\mathfrak{f}$ and the toll of cost function $\mathfrak{C}$

First thing to consider, given a labeled data set $\{(\vec{x}_j, \vec{y}_j) : j = 1, \ldots, N\}$, if there is a representation function $\mathfrak{f}$ such that $\mathfrak{f}(\vec{x}_j) = \vec{y}_j$ for all $j = 1, \ldots, N$. This important step is often overlooked in practice. In theory, if there is

$$\vec{x}_j = \vec{x}_k \qquad \text{such that} \qquad \vec{y}_j \neq \vec{y}_k \tag{4}$$

then no such function exist. In other words $\mathfrak{f}$ is a function of more variables than provided in the data set and the idea approximation by $\tilde{f}$ is meaningless.

One may always assume some measurement error $\varepsilon > 0$ (noise) of the data set and consider instead a weaker condition

$$\vec{x}_j = \vec{x}_k \quad \Longrightarrow \quad \|\vec{y}_j - \vec{y}_k\|_{l^2} < \varepsilon \tag{5}$$

necessary for existence of a function $\mathfrak{f}$. In such case one may look at the average values of duplicate points

$$f_{ave} = \frac{\sum_{\vec{x}_j = \vec{x}_k} \vec{y}_k}{\sum_{\vec{x}_j = \vec{x}_k} 1} \tag{6}$$

and choose to approximate f_{ave} instead of $\mathfrak{f}$. There is no guarantee that a good approximation $\tilde{f}$ of function f_{ave} is a good approximation of $\mathfrak{f}$ itself.

One should also consider the norm $\|\cdot\|_d$ of the approximation function $\mathfrak{f}$. It is well known that various classes of "nice" functions are dense in L^p spaces. In particular, the class of functions $\tilde{f}$ defined in (3) are dense with respect to convergence in measure and L^p norm (see [1]). This important fact implies that given any functions $\mathfrak{f}$ and any $\varepsilon > 0$, there is a function $\tilde{f}$ s.t.

$$\left\|\mathfrak{f} - \tilde{f}\right\|_{L^p} < \varepsilon. \tag{7}$$

The approximation function $\tilde{f}$ is a function of weights vectors $\vec{w}_i$. The pursuit of such function $\tilde{f}$ is a two part problem. The first part, defining the structure of the neural network, is done by a human. The methodology behind the choice of NN structures is at the stage of experimental science. The second part, weights optimization, is done by a computer, usually capable of trillions of operations per second. Needless to say that the effectiveness of latter part depends heavily on the former.

[1] We shall return to the importance of this condition later to discuss the design and structure of NN

In practice one usually does not use an approximation with respect to L^p norm. Computationally one may only evaluate the function $\mathfrak{f}$ at finitely many points and approximate it by $\tilde{f}$ at such points, often with respect to the $\|\cdot\|_{l^p}$ norm. Since for all $0 < p < q < \infty$,

$$\left\|\mathfrak{f}(y_j) - \tilde{f}(y_j)\right\|_{l^q} \leqslant \left\|\mathfrak{f}(y_j) - \tilde{f}(y_j)\right\|_{l^p}, \tag{8}$$

one can choose any $p > 0$ to obtain the approximation for all $q \geqslant p$. [1]

Here is an interesting question. Given $f \in L^p(\mathbb{R}^k)$, does it follow that the sequence $\{f(\vec{y}_j)\}_{j=1}^{\infty} \in l^p$ for any choice $\vec{y}_j \in \mathbb{R}^k$? One can easily show it is not the case. What about a sequence of randomly chosen points $\vec{y}_j \in \mathbb{R}^k$? In this case the answer is affirmative.

What about the converse statement? Given a sequence $\{f(\vec{y}_j)\}_{j=1}^{\infty} \in l^p$, does it follow that $f \in L^p(\mathbb{R}^k)$? What if for any sequence of points $\{\vec{y}_j\}_{j=1}^{\infty}$ in the domain of f, the sequence $\{f(\vec{y}_j)\}_{j=1}^{\infty}$ belongs to l^p? What if the measure μ of the domain $\mathcal{D}$ of $\mathfrak{f}$ is not a Lebesgue measure?

Things get even more bizarre if the set function μ is only finitely additive. In some cases L^p spaces may not be complete for any $p > 0$. In Measure Theory, "functions" that agree almost everywhere are indistinguishable. In case of finitely additive measures the equivalence classes of a "function" are often more complex. Why would anyone care about finitely (and not countably) additive measure on $\mathcal{D}$? From the point of view of Constructive Mathematics, it is impossible to verify countable additivity of μ in the first place.

3 Some Considerations of feed forward neural networks

Using a gradient descent method, also known as back-propagation, to optimize weights $\vec{w}_j$, one obtains critical points of the function $\mathfrak{C} = \left\|\mathfrak{f} - \tilde{f}\right\|_d$. Statistically speaking, saddle points in $\mathbb{R}^n$ are more probable than maxima and minima if $n > 2$. Thus clever enough gradient descent algorithm will not yield a saddle point. There is no guaranty that the function $\mathfrak{C}$ does not have more than one local minimum, in which case such algorithm may converge to a local instead of the global minimum of $\mathfrak{C}$.

It is also important to know weather a gradient descent algorithm converges to a local minimum of $\mathfrak{C}$. One may think that should not be a problem. Unfortunately most variable learning rate algorithms (keras optimizers) are designed to increase the convergence rate without a guarantee of asymptotic convergence to a local minimum. The are a few computational problems with such algorithms. One problem comes from the fact that all weights are updated simultaneously after each instance (epoch) which may cause the subsequent set of weights to yield a larger value of $\mathfrak{C}$. Another problem arises from the choice of the learning rate $\|\triangle\vec{w}_j\|_2$ for each instance. It is often proportional to the absolute value of the

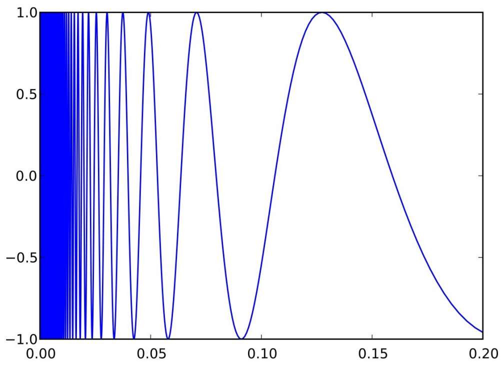
[1] Since for any sequence $x = \{x_i\}_{i\in\mathbb{N}}$ of complex numbers x_i, $\|x\|_{l^2} \leqslant \|x\|_{l^1}$, *"l1 weights regularization"* is also an *"l2 regularization"*, so *"(l1 and l2)-regularization"* is redundant.

gradient of P_j. If function $\mathfrak{C}$ is concave up near local minimum, the value $\|\triangle\vec{w}_j\|_2$ is likely to be too large.

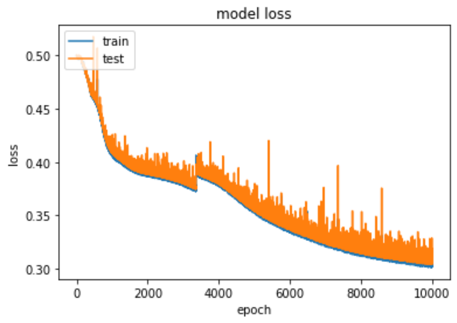
Another common practice in supervised machine learning is to partition given labeled data set into the training and validation subsets. The validation set is only used to estimate the accuracy of the model at each stage. The goal of the algorithm is to minimize both training and validation error. This idea implicitly assumes some similarity of the function $\mathfrak{f}$ on these two sets.

Assume, for example, we are trying to approximate the function

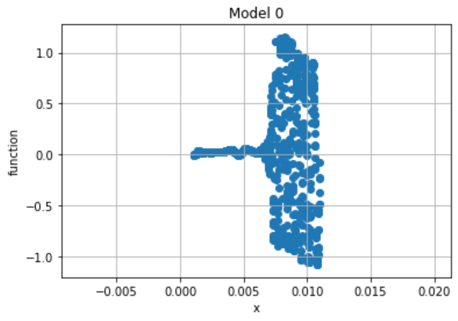
$$\mathfrak{f}(x) = \sin\left(\frac{1}{x}\right), \qquad x > 0 \tag{9}$$



Try to approximate this function using a neural network of any size so that the mean square error on 10^3 random points of the interval $(0, 0.01)$ is less than 0.1. Using Tensorflow 2 with four fully connected layers of size 800, activation function LeakyReLU(alpha=0.01), on 10^5 randomly generated points in the interval $(0.001, 0.011)$ after 10^4 epochs with Adam optimizer and validation split 0.2 I obtained the following approximation and mean square error.



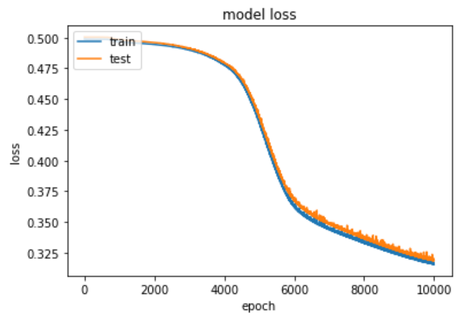
(a)



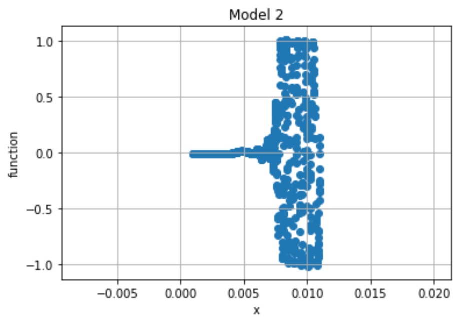
(b)

Figure 1: Model History and Evaluation (a) and (b).

With with sixteen fully connected layers of size 200 and other parameters unchanged, the approximation and mean square error are as follows.



(a)



(b)

Figure 2: Model History and Evaluation (a) and (b).

The following example is somewhat challenging, but even more pathological.

Example 3.1. One can construct a measurable subset S of $[0,1]$ with the following properties. Both S and its complement, S^c, in $[0,1]$ are totally disconnected (contain no interval) and

$$\mu(S) = \mu(S^c) = \frac{1}{2}. \tag{10}$$

Let $\mathfrak{f}$ be the characteristic function of such set,

$$\mathfrak{f}(x) = I_S(x), \qquad x \in [0,1]. \tag{11}$$

Approximation of $\mathfrak{f}$ by a function $\tilde{f}$ in (3) on any infinite countable subset of $[0,1]$ with respect to l^2 norm is computationally impossible because for each $x \in [0,1]$ the probability of $\{f(x) = 0\}$ is equal the probability of $\{f(x) = 1\}$.

Each domain $\mathcal{D}$ of the above examples is one dimensional, and it is well known that approximation problem becomes more challenging as the dimension increases.

4 Simple functions and Adaptive Neural Networks

Let us take another look at the machine learning problem of approximation of a function $\mathfrak{f}$ given its values at a countable set of points $S = \{\vec{x}_1, \vec{x}_2, ...\} \subseteq \mathcal{D} \subseteq \mathbb{R}^k$. If we only assume that $\mathfrak{f} \in L^p(\mathcal{D})$, it may be impossible to approximate $\mathfrak{f}$ on S with respect to mean square error (see example 3.1). On the other hand, if we assume some smoothness conditions (like bounded variation), to predict the value of $\mathfrak{f}$ at $\vec{x} \in \mathcal{D} \setminus S$, one could take the average of values of $\mathfrak{f}$ in S in some neighborhood of $\vec{x}$. In such case, do we really need deep neural networks to construct $\tilde{f}$ approximation of $\mathfrak{f}$, or is it just an excuse to own the latest Nvidia graphics card?

Note that the current methods in neural networks are using approximation of a given measurable function $\mathfrak{f}$ by almost everywhere continuous function $\tilde{f}$. In Measure Theory one first approximates a measurable function $\mathfrak{f}$ by simple functions. The whole point of Lebesgue integration is to use simple functions instead of piece-wise continuous (step) functions. Neural networks representing simple functions would not consume as much computational power required by matrix multiplication. Some work in this direction [2] is known as Lookup Tables, but no connection between simple functions and lookup tables have been made explicitly.

Another interesting direction would be an algorithm which designs the structure of a neural network. To implement this approach, such algorithm would have to control the width of layers and the depth of the network.

To address the width control, one could partition each layer, i, of neurons into two subsets, A_i and B_i, with zero weights of neurons in A_i, and non-zero weights of neurons in B_i. Here is where the condition

$$P_i(\vec{0}) = 0 \tag{12}$$

of an activation function P_i becomes significant. If (12) holds, the neurons from A_i contribute nothing to the value of $\tilde{f}$. One could think of neurons in A_i as auxiliary neurons. As long as weight optimization algorithm does not compute gradients of neurons in A_i, the computational complexity of the network is equivalent to one containing only the neurons in B_i's.

To adapt the depth of a neural network, one could partition all layers of neurons into two sets, S and T, such that the layers in S precede the layers in T. If all but one neurons in layer i from T belongs to A_i, and the remaining neuron in B_i assigned $\vec{w}_i$ consisting of ones, as long as $P_i(x) = x$, such layer acts as an identity function. If one does not compute the gradients and update the values $\vec{w}_i$ for layer i in T, the computational burden of such layers is negligible.

In other words, the layers in set T are dormant and act as the identity function and each neuron that belongs to set A_i acts as a place holder. Increasing the size of A_i's and size T would increases the maximum approximation accuracy with little increase in computations.

One could then implement a rule for a "switch" of a layer of neurons in T to a layer in S based on a threshold of the gradient of $\mathfrak{C}$. One could similarly implement a rule for a "switch" of neurons from A_i to B_i. Implementation of such switches automates the growth of width and depth of neural networks to accommodate the approximation accuracy without any human interference.

The above described adaptive algorithm is somewhat similar to a learning process of an adult human brain. Such brain contains constant number of neurons, but the number of neural connections is changing while learning a new skill. One could also consider a reverse switch from class B_i to A_i to implement the ability to "forget" no longer needed skill. Going a step further, one could allow the neurons in each A_i to be shared by multiple networks working in parallel.

5 Computer Vision

One of the promising areas of neural network application is so called computer vision. In recent years convolutional neural networks had a significant progress in object detection and recognition from images and video data. One of the challenges of object classification is a consequence of so-called "curse of dimensionality". Each $m \times n$ pixels image of an object is often treated as $(m \cdot n)$-dimensional vector. Even a moderate size picture of 1024×1024 pixels without some reprocessing presents a computational challenge for modern machines

There is another problem with the idea of representing $m \times n$ pixels images as $(m \cdot n)$-dimensional vectors. By converting 2D objects into vectors, one loses the internal structure of the underlying Cartesian space. Assume, for example, we have a gray scale $n \times n$ pixel image of a single object O. Let $f(x, y) \in [0, 1]$ be the grayness intensity value the pixel with Cartesian coordinates (x, y), with $x, y = 1, 2, ..., n$. If $\varepsilon > 0$ is small, one can usually assume that functions

$$f(x \pm 1, y), \quad f(x, y \pm 1), \quad f(n - x, y), \quad f(x, n - y), \quad f(x - 1, y), \quad f(x, y) \pm \varepsilon \tag{13}$$

which correspond to shifts, reflections and intensity change, would also represent the same object O. One could similarly define a small rotation and noise invariance of the function representation of the given object. Enforcing such invariance rules on the structure of neural networks is far more difficult.

We think of objects as three dimensional, so one could assume that dimension of the solution space of object classification should be of same order of magnitude. Perhaps some difficulties of object classification follow from the complexity of equivalence relation defining each class of objects. Often times such relation is not based only on the three dimensional space. How, for example, would you recognize a bottle? Since bottles come in various shapes sizes, even if we had a rigorous definition of shape, equivalence rule of the "bottle" class would be complicated. Yet, when an adult human is presented with a previously unseen and even unusual bottle, would recognise it without much effort. Our notion of bottles comes from their extensive use in daily life. It seems unlikely that the problem of computer vision would have a computationally feasible solution by a narrow AI algorithm trained only on images.

6 Recurrent Neural Networks

Another class of networks, called recurrent NNs, are designed to predict the n-the value, $\vec{y}_n$, of a sequence of vectors $\{\vec{y}_n\}_{n \in \mathbb{N}}$, given all previous values $\vec{y}_1, \ldots, \vec{y}_{n-1}$. One would think this type of problem requires different approach, yet the common practice is to modify the connections of feed forward neural network and use good old gradient descent.

Fourier series was fist thing came to my mind when looking at the above problem. If the sequence is periodic we would discover this fact within two periods of the sequence.

Since not all functions are periodic, one could next assume the sequence function is almost periodic. For example, the class of Besicovich almost periodic functions on $\mathbb{C}$ consist of trigonometric polynomials of the form

$$P(x) = \sum_{k=1}^{n} a(\eta_k) e^{i\eta_k x}, \qquad \eta_k \in \mathbb{R}, \quad a(\eta_k; P) \in \mathbb{C} \tag{14}$$

and their completion with the norm

$$\|P\|_{B^2_{ap}} = \lim_{\tau\to\infty} \left(\frac{1}{2\tau} \int_{-\tau}^{\tau} |P(x)|^2 dx \right)^{1/2}. \tag{15}$$

This is a large set of functions that need not be periodic. Even very easy almost periodic function $f(x) = e^{ix} + e^{i\sqrt{2}x}$ is not periodic. It would be interesting to use recurrent neural networks to approximate this function.

One can easily generalize Fourier series to this class of functions and use a computational power to estimate the Fourier series instead of using gradient descent. Since

$$\langle e^{i\eta_j x}, e^{i\eta_k x} \rangle = \lim_{\tau\to\infty} \left(\frac{1}{2\tau} \int_{-\tau}^{\tau} e^{i\eta_j x} \cdot e^{-i\eta_k x} dx \right) = \begin{cases} 1 & \eta_j = \eta_k \\ 0 & \eta_j \neq \eta_k \end{cases} \tag{16}$$

the set of functions functions $\{e^{i\eta_k x}\}_{k\in\mathbb{N}}$ form an orthonormal system and the completion of trigonometric polynomials in (14) is a Hilbert space. One may compute generalized Fourier coefficients of a Besicovich almost periodic function f using

$$a(\eta; f) = \lim_{\tau\to\infty} \left(\frac{1}{2\tau} \int_{-\tau}^{\tau} f(x) e^{-i\eta x} dx \right), \qquad \eta \in \mathbb{R}. \tag{17}$$

Moreover, one can take advantage of Harmonic Analysis theory, by first defining a finitely additive measure γ with

$$\gamma(S) = \lim_{\tau\to\infty} \left(\frac{1}{2\tau} \int_{(-\tau,\tau)\cap S} 1\, dx \right), \qquad S \subseteq \mathbb{R} \tag{18}$$

and then obtaining $\|\cdot\|_{B^2_{ap}}$ in (15) as the usual L^2 norm with the measure γ.

Only recently Fourier series were used in the new design of recurrent neural network called "Transformers" introduced by authors of the paper "Attention is all you need" [3].

References

[1] K Hornik, M. Stinchcombe, H. White: Multilayer Feedforward Networks are Universal Approximators *Neural Networks* Vol 2, pp. 359 - 366, 1989

[2] R. Isermann , M. Münchhof M: Neural Networks and Lookup Tables for Identification *Identification of Dynamic Systems* Springer, Berlin, Heidelberg, 2011

[3] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. Gomez, L. Kaiser, I. Plosukhin: Attention is All You Need *31st Conference on Neural Information Processing Systems* Long Beach, CA, USA, 2017