How the fundamental concepts of mathematics and physics explain deep learning

Preprint · November 2018

Jean Thierry-Mieg
National Institutes of Health

How the fundamental concepts of mathematics and physics explain deep learning.

Jean Thierry-Mieg

NCBI, National Library of Medicine, National Institutes of Health,
8600 Rockville Pike, Bethesda MD20894, USA.
E-mail: mieg@ncbi.nlm.nih.gov

Abstract

Starting from the Fermat's principle of least action, which governs classical and quantum mechanics and from the theory of exterior differential forms, which governs the geometry of curved manifolds, we show how to derive the equations governing neural networks in an intrinsic, coordinate invariant way, where the loss function plays the role of the Hamitonian. To be covariant, these equations imply a layer metric which is instrumental in pretraining and explains the role of conjugation when using complex numbers. The differential formalism also clarifies the relation of the gradient descent optimizer with Aristotelian and Newtonian mechanics and why large learning steps break the logic of the linearization procedure. We hope that this formal presentation of the differential geometry of neural networks will encourage some physicists to dive into deep learning, and reciprocally, that the specialists of deep learning will better appreciate the close interconnection of their subject with the foundations of classical and quantum field theory.

1 Background

1.1 What is a neural-net good for

The purpose of a neural-network (NN), the logical architecture behind deep learning [1], is to transform an input vector X into a labeling vector $\widehat{Y}$, for example, in a supervised learning problem, the vector X may represent an image and the output $\widehat{Y}$ a classification probability like 'this image has a probability 92/100 of representing a cat'. In more complex settings like reinforcement learning and adversarial networks [2, 3], the $\widehat{Y}$ may represent a choice between several actions. But in all cases, during the training phase, one provides a set of m training vectors $\{X\}$, computes the corresponding set of output vectors $\{\widehat{Y}\}$ and compares $\{\widehat{Y}\}$ to a set of truth vectors $\{Y\}$ representing the desired outputs of the NN. The comparison is performed by selecting a loss function

$$\mathcal{L} = \frac{1}{m} \sum_X \mathcal{L}(X) \tag{1}$$

$\mathcal{L}$ plays the role of the Hamiltonian in classical mechanics and will be used to define the time-flow of the neural network during the training iterations.

1.2 How is it designed

A neural net consists of a large collection of very simple interconnected computing cells described below, called artificial neurons. Each neuron acts nearly trivially, but complex learning emerges from their combination. The actual design of a NN is specified by a set of choices, called the hyper-parameters of the net. They include the number of layers of the net, the types and number of neurons in each layer, and their connectivity. The hyper-parameters are selected by the user and are not automatically adjustable. At present, the design of the network remains an art, but it has been observed that deep nets, with many layers, learn better than shallow nets. In 2017, many publications described NN with over 100 layers. Hence the re-branding after 2007 of the 'Artificial Neural Nets' paradigm of the 80's under the new name 'Deep learning'.

1.3 How does it work

The magic of the NN is that during the training phase, the NN modifies automatically the response of its individual neurons until the predicted outputs $\{\widehat{Y}\}$ closely match the desired outputs $\{Y\}$. Here is how it works.

Each neuron performs an affine transformation $Y = WX + b$ on its input vector X, followed by a nonlinear activation function Φ like a sigmoid, or a rectified linear unit (ReLU). The $Z = \Phi(Y)$ are then used as the X input of the next layer. The presence of several layers of nonlinearities will allow the NN to learn to recognize complex relationships in the input data, up to understanding natural language, playing chess and go, or driving a car. The basic method is amazingly simple.

At time zero, the W, b coefficients, collectively called the parameters of the NN, are initialized with small random numbers in order to break any existing symmetry. Then the time evolution of the parameters is driven by a simple differential equation, called the steepest descent, which depends on the choice of the loss function $\mathcal{L}$.

To construct this equation, the main idea is to realize that $\mathcal{L}$ is not supposed to constrain the $\{X\}$ vectors, which represent the external data, for example texts or images, but should rather be regarded as a function of the parameters $\{W\}$ of the NN, and that this dependency survives the averaging over the X

$$\mathcal{L}(W) = \frac{1}{m} \sum_X \mathcal{L}(X, W) \tag{2}$$

We now remember that we want the W to evolve in time until the network is well-trained, so we postulate that the W are unknown functions of time:

$$W = W(t). \tag{3}$$

Hence the loss function itself becomes a function of time, hopefully converging towards a global minimum:

$$\mathcal{L}(t) = \mathcal{L}(W(t)). \tag{4}$$

Let us now compute the exterior differential of $\mathcal{L}$

$$d\mathcal{L} = \frac{\partial \mathcal{L}}{\partial t}\, dt = \sum_W \frac{\partial \mathcal{L}}{\partial W}\, \frac{\partial W}{\partial t}\, dt. \tag{5}$$

Starting from the differential of the loss function, one now postulates that the time evolution of the W parameters is governed by the differential equation

$$dW = -\frac{\partial \mathcal{L}}{\partial W}\, \eta\, dt, \tag{6}$$

where the parameter η is called the learning rate [4]. The rationale for postulating this equation is explained in the next section.

2 Neural nets from the point of view of differential geometry

2.1 Fermat's principle of least action

The governing principle of a very large part of theoretical physics, including general relativity, classical and quantum mechanics, and the standard model of the fundamental interactions, is a suitable generalization of Fermat's principle of least action. In its original form, in the seventeenth century, it simply stated that a ray of light will follow the fastest path between 2 points, explaining refraction by assuming that light travels slower in water than in the air, a true statement which was verified experimentally only much later and was in plain contradiction with the unfortunate hypothesis of Descartes that light would travel faster in water than in the air. Around 1930, following Elie Cartan, Einstein and Hermann Weyl, it had become apparent that the best formalism to express the least action principle is the formalism of exterior differential geometry, whereby particles travel along straight lines, called geodesics, in a curved space representing the presence of external forces, like electromagnetism or gravity, and the eventual existence of constraints.

We would like to show that the training of a neural network follows the same paradigm and can be expressed in the same formalism. This is not really new or surprising, but this point of view is not emphasized in the recent book by Goodfellow, Bengio and Courville [2], nor in the book of Géron [3], nor in the excellent lectures of Ng [5].

The NN steepest descent equation implements Fermat's principle of least action in the following sense. The NN flows along the shortest path in parameters space leading to a given decrease of the loss function. However, to define a distance in $\{W\}$ space, we need a metric g. If we call $W^{[i]}$ an individual parameter, for example, a matrix coefficient associated to the i^{th} layer, the NN equation reads:

$$dW^{[i]} = -g^{[ij]} \frac{\partial \mathcal{L}}{\partial W^{[j]}}\, \eta\, dt. \tag{7}$$

Notice the presence in this equation of upper and lower indices, respectively called covariant and contravariant, which are needed each time one wishes to write consistent equations in a system of coordinates which is not orthonormal, i.e. either the axes are not orthogonal or the base vectors have different lengths.

The layer metric g is needed in two equations. It is needed to transform the partial derivative $\partial_{[j]}\mathcal{L}$ with a lower (contravariant) $[j]$ index into a quantity with an upper (covariant) $[i]$ index, so that it can be added to the upper (covariant) index differential $dW^{[i]}$. The g metric is also needed to construct in $\{W\}$ space the elementary square distance ds^2, familiar from general relativity:

$$ds^2 = g_{[ij]} dW^{[i]} dW^{[j]}, \tag{8}$$

where the lower (contravariant) index metric $g_{[ij]}$ is defined as the inverse of the upper (covariant) index metric $g^{[ij]}$:

$$g^{[ij]} g_{[jk]} = \delta^{[i]}_{[k]}. \tag{9}$$

If we recognize that at each layer W, and hence dW, is a matrix, the natural extension is the Frobenius norm:

$$ds^2 = g_{[ij]} \;\; Tr(dW^{[i]t} \, dW^{[j]}), \tag{10}$$

which is simply the sum of the squares of all elements of the dW matrix. weighted by the g metric. The NN equation then implies that the length of the path from an initial configuration W_0 to a final configuration W_1 computed as

$$I = \int_{W_0}^{W_1} ds \tag{11}$$

is minimal relative to the distance from W_0 to any other (local) configuration W with the same loss function as W_1, exactly as required by the principle of least action [6]. At each instant, the W flow normally to the sheet of configurations with equal loss, where orthogonality is defined relative to the metric g.

The existence of the layer metric g is implied by the structure of the equations. But from a pragmatic point of view, it plays a useful role. Its meaning is that all cell layers do not have to be created equal. A classical method introduced by Hinton is called pretraining. One first trains a rather shallow NN on a large set of unlabeled examples, allowing the NN to recognize the main features of a new kind of data, and then one freezes the coefficients of these layers and trains additional layers which try to transform the output of the shallow network into the desired results using a possibly smaller set of labeled examples with known truth values.

For example, suppose that the first 6 layers of the network were pretrained and that 3 additional layers need training. We could set $g^{[ij]} = 0$ for $i, j <= 6$ but $g^{[ij]} = 1$ for $i, j > 6$. This is equivalent to setting $g_{[ij]} = \infty$ for $i, j <= 6$ and makes the parameters of the low layers immutable. But we could also set the low layer $g_{[ij]}$ to a high value, like 100, allowing the pretrained part of the network to adjust conservatively to the new condition at a very slow rate. We could also decide that the parameters of layers with many cells are stiffer or softer than the parameters of layers with fewer cells. Such techniques are widely used in pretraining deep networks.

Here we have treated the metric as layered, but if a layer contains several distinct types of cells, it would also make sense to give a different stiffness to each group.

We see that writing the equations of the neural net in the classic notations of the physicist forced us to introduce a metric and to anticipate the concept of variable stiffness of the successive layers of the NN.

2.2 Understanding the metric when using complex numbers

A way to illustrate the role of the g metric is to analyze the situation where the W coefficients are complex numbers. The square length of a complex number $z = x + iy$ is not given by the square of z but by the product of z by its conjugate $\overline{z}$. In other words, in z, $\overline{z}$ space, the metric is anti-diagonal

$$g_{z\,z} = g_{\overline{z}\,\overline{z}} = 0 \qquad g_{z\,\overline{z}} = g_{\overline{z}\,z} = 1/2 \tag{12}$$

As a result, we find that the differential of W is proportional to the derivative of $\mathcal{L}$ with respect to $\overline{W}$ rather than with respect to W because the g metric always couples a complex to its conjugate:

$$dW^{[i]} = -2\, g^{[ij]} \frac{\partial \mathcal{L}}{\partial \overline{W^{[j]}}}\, \eta\, dt. \tag{13}$$

The need to take the partial derivatives with respect to the complex conjugates of the parameters would not be self-evident if we had not explicitly introduced the g metric.

Complex neural networks are naturally important in domains where the input vectors X are best described by complex functions, as in sound recognition or imaging where the phase of the signal characterizes the direction of the source. But they are also promising in other domains. The complex differentiable (holomorphic) functions are way more constrained than real differentiable functions, and the space of vectors of norm one ($z\overline{z} = 1$) is connected in the complex case $z = e^{i\phi}$, but not on the real lane $1, -1$. These two properties facilitate the exploration of the parameter landscape. See [7] for a recent application of complex neural nets to the analysis of MRI medical pictures, [8] for an application to sound patterns, or [9] for an introduction to the complex Cayley transform.

2.3 Pullback of differential forms

We show in this section that the NN differential equation and the NN paradigm of back-propagation have a very simple an elegant interpretation in terms of differential forms [10].

Let us first remember a 'sweet technicality'. The fundamental property of a differential increment, usually denoted dt, is that it is a true infinitesimal object. In any equation, terms with zero, one or a product of two d are independent of each other. For example

$$u + vdt = 0 \Leftrightarrow u = 0, v = 0. \tag{14}$$

As a result, all equations in dt are linearized, hence easy to solve.

Let us now introduce the geometrical notions of tangent vectors and differential forms. Consider a surface with local coordinates x and y, for example the surface S of a sphere. Given a real valued function $f(x,y)$, the infinitesimal variation of a variable $x \to x+\epsilon$ induces a correlative variation of the function $f(x) \to f(x) + \epsilon \frac{\partial f}{\partial x}$. This variation is called the tangent action f_* of f on the tangent vector $\epsilon \frac{\partial}{\partial x}$. The 2 tangent vectors $\frac{\partial}{\partial x}$ and $\frac{\partial}{\partial y}$ can be visualized as forming the basis of the plane tangent to the surface at position (x,y). One then defines the differential forms (dx, dy) as their dual vectors relative to the scalar product $< \partial x, dx > = 1$, $< \partial x, dy > = 0$, $< \partial y, dx > = 0$, $< \partial y, dy > = 1$, and the exterior differential df of f as $df = \frac{\partial f}{\partial x} dx + \frac{\partial f}{\partial y} dy$. (dx, dy) span the plane cotangent to S. This may seem quite formal, but the advantage of df is that it is intrinsic, meaning that it no longer depends on the choice of the coordinate system on the surface S. Suppose we change coordinates and use $X = 2x$, the partial derivative of f is divided by two: $\partial f/\partial X = \frac{1}{2} \partial f/\partial x$, but since $dX = 2dx$ we have $df = \partial f/\partial x \; dx = \partial f/\partial X \; dX$. This is also true if we rotate the coordinates (x,y) into one another using a matrix multiplication: df remains invariant.

Suppose now that the point x, y moves with time along some line on the surface, so x, y and f become functions of time $x(t)$, $y(t)$ and $f(x(t), y(t))$. Consider time itself as a 1-dimensional manifold T, i.e. a line, with its own tangent vector $\frac{\partial}{\partial t}$ and its own differential form dt. Then, the mapping $x(t)$ maps t to $x(t)$ and induces a mapping x_* of the tangent vector $\frac{\partial}{\partial t}$ tangent to T to a vector tangent to the surface (x,y)

$$\begin{array}{c} x_* : T_* \to S_* \\ \frac{\partial}{\partial t} \to \frac{\partial x}{\partial t} \frac{\partial}{\partial x} \end{array} \tag{15}$$

Since $\frac{\partial}{\partial t}$ has been transported to a vector tangent to S, we can compute its scalar product with any differential form cotangent to S. In other words, the differential forms over S have been mapped back to differential forms over T so that their images are dual to a vector tangent to T. This reciprocal map is called the cotangent application or pullback of the application x. It is defined by the linear completion of the equation

$$x^*(dx) = \frac{\partial x}{\partial t} \, dt \tag{16}$$

hence

$$x^*(df) = \frac{\partial f}{\partial x} \, \frac{\partial x}{\partial t} \, dt \tag{17}$$

As can be observed in the last equation, the pullback, as the name indicates, works backwards. If we compose 2 applications f and g we have

$$\begin{array}{rl} f \ : & E \to F \\ g \ : & F \to G \\ g \circ f \ : & E \to G \\ (g \circ f)^* = f^* \circ g^*. & \end{array} \tag{18}$$

If f is represented by a matrix, the tangent application f_*, which maps E_* on F_*, is represented by the matrix of partial derivatives, i.e. by the Jacobian of f, and the cotangent application f^* by the transpose of f_*, i.e. the transpose of the Jacobian. Coming back to the study of the NN, we observe that the time pull back of the differential of the loss funtion

$$T^*(d\mathcal{L}) = \frac{\partial \mathcal{L}}{\partial W} \frac{\partial W}{\partial t} \, dt \tag{19}$$

defines a differential form $dW = \frac{\partial W}{\partial t} \, dt$ cotangent to the parameter space. But dW does not specify a tangent direction. In the colorful terminology of Wheeler , a form is not a direction of displacement but 'a machine to produce a number out of a displacement' [11] page 115, his way of saying that a form is an element of the dual of the tangent vectors. So to define a direction maximizing the time evolution of the loss function, in accordance to the principle of Fermat, we need again to introduce a metric and write the time derivative of the loss function as a scalar product:

$$\Delta = \frac{\partial \mathcal{L}}{\partial t} =< \frac{\partial \mathcal{L}}{\partial W}, \frac{\partial W}{\partial t} >= g^{[ij]} \frac{\partial \mathcal{L}}{\partial W^{[i]}} \frac{\partial W_{[j]}}{\partial t} \tag{20}$$

Since we are only interested in a choice of direction, we maximize Δ under the constraint that $(\frac{\partial W}{\partial t})^2$ is constant. Any component of $\frac{\partial W}{\partial t}$ perpendicular to $\frac{\partial \mathcal{L}}{\partial W}$ increases $(\frac{\partial W}{\partial t})^2$ without modifying Δ. It follows that $\frac{\partial W}{\partial t}$ must be proportional to $\frac{\partial \mathcal{L}}{\partial W}$:

$$\frac{\partial W^{[i]}}{\partial t} = -\eta \ \ (g^{[ij]} \frac{\partial \mathcal{L}}{\partial W^{[j]}}) \tag{21}$$

which is equivalent to equation (7). The present derivation shows that η does not have to be a constant. It can be an arbitrary function of time. For example, we could choose $\eta(t) = \mathcal{L}(t)$, which would be equivalent to squaring the loss function:

$$\eta(t) = \mathcal{L}(t) \qquad \Rightarrow \qquad \frac{\partial W}{\partial t} = -\mathcal{L} \, \frac{\partial \mathcal{L}}{\partial W} = -\frac{1}{2} \, \frac{\partial \mathcal{L}^2}{\partial W} \tag{22}$$

When the function $\eta(t)$ is modified, the speed of evolution of the W parameters is modified but they follow the same trajectories and converge to the same minimum because, in the differential formalism, the product $\eta \, dt$ remains infinitesimal even if η is large. In deep learning, changing η in a clever way is called a choice of 'learning rate policy'.

2.4 Mechanical interpretation of the gradient descent optimizers

The loss function $\mathcal{L}(W)$ can be interpreted as the potential energy of the system, usually denoted $V(x)$ in classical mechanics. The negative of the gradient of $\mathcal{L}$ with respect to W therefore represents the force F causing the network to move across the parameter space W with speed v. In these notations, the pullback equation reads:

$$v = \frac{\partial W}{\partial t} = \eta F. \tag{23}$$

As in Aristotle mechanics [12], this equation tells us that the speed v of the mobile is proportional to the force. This equation is physically correct only in a situation dominated by a huge friction, like a horse pulling a plough. In those cases, the motion is usually very slow. If we hope to accelerate the convergence of the network, it seems reasonable to look for an equation applicable to cases with less friction and faster displacement and to postulate with Newton that the acceleration a, rather than the speed v, is proportional to the force, according to the equation:

$$ma = m\frac{\partial v}{\partial t} = F - \lambda v \tag{24}$$

describing the acceleration a, of a point of mass m, subject to a force F, with friction coefficient λ. The mechanical inertia associated to the mass of the mobile stabilizes the module of the speed and the orientation of the trajectory. On a flat section of the landscape, where $F = 0$, the motion continues and the speed v only decays exponentially as $e^{-\lambda t/m}$. This method, introduced in [13] is called the gradient descent 'momentum' optimizer. As hoped, the network converges faster and more often than with the Aristotle equation.

The current best methods, RMS-propagation [14] and Adam [15], introduce a further refinement. Close examination of the trajectories shows that the network is subject to a Brownian motion because each new set of training examples introduces a modification of the loss function $\mathcal{L}(W)$ and tends to drive the weight configuration in a different direction [16]. However, only the average motion is desirable. The idea is to maintain a rolling average $< F >$ of the force, with exponential time decay of the previous values,

$$\frac{\partial < F >}{\partial t} = -\beta(< F > -F) \tag{25}$$

and to postulate the equation:

$$ma = m\frac{\partial v}{\partial t} = \gamma < F > -\lambda v \tag{26}$$

where the variable coefficient γ dampens the effect of the components of $< F >$ in the directions in which F fluctuates, as measured by maintaining the rolling exponential time average of F_w^2 in each w direction. These methods strongly accelerate the convergence towards a good local minimum of $\mathcal{L}$, although it is sometimes reported that the network is over adapted to the examples and does not generalizes so well to new test examples [17].

3 Remarks

3.1 On the paucity of local minima in high dimension

A network can only be trained well if the gradient descent paradigm can discover configurations with a very low loss function, such that each training example X is mapped very close to its known target value Y, furthermore hoping that such a good mapping will generalize well to new test examples not seen during the training. Therefore, a very interesting question is to evaluate the risk of being trapped in a false minimum.

Drawing from our life-long 3-dimensional experience, we expect local minima to be very frequent: in a mountain landscape, there are many lakes and on a rainy day a huge number of little puddles of water are forming. However, neural networks often have millions of W parameters, and in high dimension absolute minima become extremely rare relative to saddle points [1]. In a space of dimension D a horizontal plane tangent to an equipotential $\mathcal{L}$ surface is defined by $D-1$ linear equations, indicating that each partial derivative relative to a different direction vanishes. In each of these directions, the second derivative may point up or down, yielding 2^D configurations, but only one of them, when all second derivatives point upwards, corresponds to a local minimum. All other configurations characterize saddle points where some escape routes remain open. The true local minima are therefore exponentially rare, with probability 2^{-D}, relative to the saddle points, and this helps to understand why neural nets are not constantly trapped in false minima. Some authors even try to show that, in concrete situations, the different minima discovered in the network are most often connected by a quasi-horizontal path [18]. These qualitative observations may help understand the otherwise amazing success of gradient descent equation to find deep minima in these extremely complex manifolds. It would be interesting to know in which sense the conjecture that there would exist a single connected globally minimal region could be validated.

3.2 On the importance of the activation functions

As discussed in the introduction, a neural network, is generally composed of small computing units, the neurons, performing a linear matrix calculation, followed by a sigmoid or diode like element, called a rectified linear unit (ReLU), only allowing the propagation of signals exceeding a self-adjusting threshold. This may seem innocuous, but these diodes are critical in the decision process. As discussed long ago by Minsky and Papert [19], a linear network cannot solve a single bit exclusive-or ($a\ XOR\ b$) classification problem because the 2 classes $(0,0)$ and $(1,1)$ versus $(0,1)$ and $(1,0)$ cannot be separated by a linear equation. But using a hidden layer equipped with a ReLU, it is trivial to compute the sum $a+b$ and separate the value 1 from 0 and 2. A neural network with several nonlinear layers automatically builds a succession of higher level representations of the data up to super-human capabilities in certain well-defined problems like playing chess, go or in various classification problems. The nonlinearities do not modify our discussion of the pullback equation. However, they greatly modify the landscape and the topology of the minima, as discussed for example in [20].

3.3 On the structure of the loss function

Given the input vectors $\{X\}$ and the outputs $\{\widehat{Y}\}$, the simplest way to compare the $\{\widehat{Y}\}$ to the desired results $\{Y\}$ is to choose as loss function the Euclidean distance

$$\mathcal{L} = \frac{1}{2}\ (Y - \widehat{Y})^2 \tag{27}$$

This distance again depends on an implicit choice of metric γ_{ab}

$$\mathcal{L} = \frac{1}{2}\ \sum_{a,b} \gamma_{ab}\ \ (Y - \widehat{Y})^a\ (Y - \widehat{Y})^b \tag{28}$$

where the sum extends over all the training examples and the γ metric may be used to weigh the examples according to their importance or their correlations.

Computing the pull-back, we immediately see that the gradient is proportional to the difference $(Y - \widehat{Y})$

$$\begin{aligned} d\mathcal{L} &= (Y - \widehat{Y})^a \gamma_{ab} \tfrac{\partial \widehat{Y}^b}{\partial W^i}\ dW^i \\ dW^i &= -\eta g^{ij} \gamma_{ab} \tfrac{\partial \widehat{Y}^a}{\partial W^j}\ (\widehat{Y} - Y)^b \end{aligned} \tag{29}$$

In a classification problem, rather than predicting the values of the vector Y, it is customary to predict the probability as

$$\widehat{P}^a = \frac{e^{\widehat{Y}^a}}{\sum_b e^{\widehat{Y}^b}} \quad \rightarrow \quad \sum_a \widehat{P}^a = 1 \tag{30}$$

where $\widehat{Y}$ plays the role of the energy/temperature ratio familiar in the thermodynamics Boltzmann distribution $exp(-E/kT)$. One then selects as loss function the cross-entropy

$$\mathcal{L} = -\ \sum_{a,b}\ \gamma_{ab}\ \ P^a\ \ log(\widehat{P}^b) \tag{31}$$

a function closely related to the Shannon entropy $-\sum\ P\ log(P)$. $\mathcal{L}$ measures the distance between the desired probability distribution P and the predicted distribution $\widehat{P}$.

The magic of this choice, as can be verified by a direct calculation, is that the gradient descent equation

$$dW^i = -\eta g^{ij} \gamma_{ab} \frac{\partial \widehat{Y}^a}{\partial W^j}\ (\widehat{P} - P)^b \tag{32}$$

is nearly identical to the quadratic case, with the simple replacement of the difference $\widehat{Y} - Y$ by $\widehat{P} - P$. This extremely beautiful equation is one of the jewels of deep learning.

3.4 Two simple analytic applications

When we look at the computer programs used to train the neural networks, it may seem that they work because they use successive discrete training calculations. We show here, that the NN approach to equilibrium follows normal differential equations which, in the simplest cases, can be integrated analytically using the usual rules of calculus.

The first example is often used in NN as a regulator. It corresponds to a mass term in classical mechanics. To ensure the existence of a single global minimum, the loss function should be chosen to be convex and bounded from below. The simplest such function is the parabola,

$$\mathcal{L}(W) = \frac{1}{2} W^2 \tag{33}$$

We have

$$d\mathcal{L} = \frac{\partial \mathcal{L}}{\partial W}\, dW = W\, dW, \tag{34}$$

hence the NN differential equation can be integrated analytically

$$\begin{aligned} dW &= -W\, \eta dt, \\ \tfrac{dW}{W} &= d(Log(W)) = -\eta dt, \\ W(t) &= W_0 e^{-\eta t}, \\ \mathcal{L}(t) &= \mathcal{L}_0 e^{-2\eta t}. \end{aligned} \tag{35}$$

The parameter of the net moves continuously down the parabola, and the loss function decreases to zero in an exponential way.

The next simplest case is the quartic equation, which illustrates the fact that a softer loss function slows down the approach to equilibrium. Let us have:

$$\begin{aligned} \mathcal{L}(t) &= \tfrac{\alpha}{8} W^4(t), \\ d\mathcal{L} &= \tfrac{\alpha}{2} W^3(t) dt. \end{aligned} \tag{36}$$

The NN differential equation becomes

$$\begin{aligned} dW &= -\tfrac{\alpha}{2} W^3\, \eta\, dt \\ -2\tfrac{dW}{W^3} &= d\tfrac{1}{W^2} = \alpha\eta dt \\ \tfrac{1}{W^2} - \tfrac{1}{W_0^2} &= \alpha\eta t \\ \\ W(t) &= \tfrac{W_0}{\sqrt{1+\alpha\eta W_0^2 t}} \\ \mathcal{L}(t) &= \tfrac{\mathcal{L}_0}{(1+\alpha\eta W_0^2 t)^2} \end{aligned} \tag{37}$$

where W_0 is the arbitrary initial value of the parameter. As expected the approach to equilibrium in t^{-2} is slower than in the previous case which behaved as e^{-t}.

3.5 On the stability of the calculations

An important observation is that thanks to the linearization procedure, even in the presence of nonlinear activation functions, we never need to compute the inverse of a matrix. Rather, if in the forward propagation we compute the product of the matrices corresponding to several layers, in the pullback phase, we only need the transpose of the Jacobian, which is equal to the product of the transposed Jacobian corresponding to each layer in backwards order

$$(J^{[3]}J^{[2]}J^{[1]})^t = J^{[1]t}J^{[2]t}J^{[3]t} \tag{38}$$

This is crucial, because a NN often involves very large matrices, and computing the inverse of a very large matrix is at best very slow and very often numerically unstable.

3.6 On the difficulties arising from the discrete nature of the computer

On a computer, we can only deal with a finite number of steps of calculation, so we must replace the infinitesimal differential equation by the approximate finite difference equation

$$\delta W = -\frac{\partial \mathcal{L}}{\partial W}\,\eta\;\delta t. \tag{39}$$

where η is the learning rate and $\eta\delta t$ now represents a small but finite quantity called the learning step. If the step is too small, one needs too many iterations, if too big, the linearization approximation may be broken since some terms of order $(\eta\delta t)^2$ may become as large or larger than some terms linear in $\eta\delta t$. These nonlinearities interfere with the logic of the calculation which may become unstable and miss the true minimum. Of course, following the classical Runge-Kutta methods dating back to 1900, it is recommended to adapt the step to the steepness of the differential equation and go fast in shallow regions and slow over cliffs. One can also introduce inertia, in the form of momenta, and encourage the W to move more consistently in well-defined directions. However, it must be understood that the main cause of the problem is not the excessive step δW in one of the D directions, where D is the number of parameters, but the possible interferences between the $D^2/2$ pairs of variables, the $D^3/6$ triplets, and so on, interferences which do not exist in the truly infinitesimal dW formalism. The problem is well illustrated by the model of a car driving on a multi-lanes freeway. Using differential equations, the car may continuously adapt its direction and follow it own lane, but if it moves by quantum jumps, it may well in a bend change lane and end up on an exit ramp, away from its final destination.

4 Conclusion

The purpose of this note was to clarify the training paradigm of a neural net using the standard concepts and notations of differential geometry and classical mechanics, a point of view not emphasized in the recent books of Goodfellow, Bengio and Courville [2], or Géron [3], or the lectures of

Ng [5]. We have shown that the neural net steepest descent equation implements Fermat's principle of least action using the cotangent pullback of the differential of the loss function. Since, as the name implies, the cotangent pullback of a differential form uses the functions describing each layer in reverse order, the back-propagation paradigm of the NN is easily understood. We have also shown that to be covariant, the equations automatically imply a layer metric which is instrumental in the pretraining of neural nets and opens the possibility of working with all kinds of numbers. In particular, if we use complex numbers, the metric introduces an otherwise mysterious complex conjugation in the back-propagation equation. The mechanical interpretation of the loss function as the potential energy of the network in parameter space helps to understand why the 'momentum' method describes a Newtonian system with less friction than the simple gradient descent equation, and clarifies the Brownian motion aspects of the current best optimizers, RMS-prop and Adam.

We also pointed out that the linearization procedure, implicit in any differential variation, avoids the calculation of inverse matrices, greatly facilitating the implementation of the neural network algorithms, but that the finite steps δt used on the computer will break the linearization logic when δt is too large because some quadratic terms proportional to δt^2 may become larger than some terms linear in δt. Finally, we recalled the beautiful interplay between the Boltzmann thermic partition function $exp(-E/kT)$ and the choice of the cross-entropy loss function $-P\ log(\widehat{P})$ leading to a gradient directly proportional to $\widehat{P} - P$.

We hope that this formal presentation of the differential geometry of the neural networks will help some physicists to dive into deep learning, and reciprocally, that the specialists of deep learning with a background in biology or computer science will better appreciate the close interconnection of their subject with the very rich literature on classical and quantum field theory, in the hope that some of the latter techniques are still awaiting to be transposed into Deep Learning.

Acknowledgments

We would like to thank Danielle Thierry-Mieg and Dominik Miketa for interesting suggestions and David Landsman for actively promoting research on neural networks. This research was supported by the Intramural Research Program of the NIH, U.S. National Library of Medicine.

References

[1] Deep Learning.
Yoshua Bengio, Yann LeCun and Geoffrey Hinton.
Nature, 521, 436-438, 2015.

[2] Deep Learning.
Ian Goodfellow, Yoshua Bengio and Aaron Courville.
MIT Press, 2016.

[3] Hands-On Machine Learning with Scikit-Learn and TensorFlow.
Aurélien Géron.
O'Reilly, 2017.

[4] Learning representations by back-propagating errors.
David E. Rumelhart, Geoffrey E. Hinton and Ronald J. Williams.
Nature. 323 (6088): 533 536, 1986.

[5] Coursera lectures on deep learning.
Andrew Ng.
https://www.coursera.org/learn/deep-neural-network, 2017.

[6] Course of Theoretical Physics, volume 1: Mechanics.
Lev Landau and Evgeny Lifshitz.
Pergamon Press Ltd. Oxford/London/Paris, 1960.

[7] Better than Real, Complex-valued neural nets for MRI fingerprinting.
Patrick Virtue and Stella X. Yu.
arXiv:1707.00070v1, 2017.

[8] Complex neural networks for audio.
Andy Sarroff.
https://andysarroff.com/papers/sarroff2018a.pdf, 2018.

[9] CayleyNets: Graph Convolutional Neural Networks with Complex Rational Spectral Filters.
Ron Levie, Federico Monti, Xavier Bresson and Michael M. Bronstein.
arXiv:1705.07664, 2017.

[10] A Comprehensive Introduction to Differential Geometry.
Michael Spivak.
Publish or Perish, INC, Houston, Texas, 1999.

[11] Gravitation.
Charles W. Misner, Kip S. Thorne and John Archibald Wheeler.
W H Freeman and Company, 2 edition, 1973.

[12] Aristotle's Physics: a Physicist's Look.
Carlo Rovelli.
arXiv:1312.4057 [physics.hist-ph], 2013.

[13] Some methods of speeding up the convergence of iteration methods.
B.T. Polyak.
USSR Computational Mathematics and Mathematical Physics,4(5):117, 1964.

[14] Divide the gradient by a running average of its recent magnitude.
Tijmen Tieleman and Geoffrey Hinton.
Lecture 6.5-rmsprop: COURSERA: Neural Networks for Machine Learning, 2012.

[15] A method for stochastic optimization.
Kingma D. and Ba J..
arXiv:1412.6980, 2014.

[16] A walk with SGD.
Chen Xing, Devansh Arpit, Christos Tsirigotis and Yoshua Bengio.
arXiv:1802.08770, 2018.

[17] Closing the Generalization Gap of Adaptive Gradient Methods in Training Deep Neural Networks.
Jinghui Chen and Quanquan Gu.
arXiv:1806.06763, 2018.

[18] Essentially No Barriers in Neural Network Energy Landscape.
Felix Draxler, Kambis Veschgini, Manfred Salmhofera and Fred A. Hamprecht.
arXiv:1803.00885, 2018.

[19] Perceptrons. An Introduction to Computational Geometry.
Marvin Minsky and Seymour Papert.
M.I.T. Press, Cambridge, Mass., 1969.

[20] Topology and Geometry of Half-Rectified Network Optimization.
C. Daniel Freeman and Joan Bruna.
arXiv:1611.01540, 2016.