Singularities in Deep Neural Networks:

A Brief Discussion about Mathematics of Deep Learning

October 28^{th}, 2020

FAPESP SPRINT Project

Title: Applications of Singularity Theory on Deep Neural Networks

Scientific Research Student: Alan Gonelli Miranda (INCTMat/CNPq fellowship)

ICMC coordinator: Raimundo N. Araújo dos Santos (SMA-ICMC)

i-PRoBe Lab / MSU coordinator: Arun Ross (MSU)

ICMC SÃO CARLOS USP

1. Introduction

1.1 Reasons for using deep networks

- Increase in performance of recognition systems due to the introduction of deep architectures for representation learning and classification;
- Crucial Properties of Deep Networks:
  - Larger number of layers as compared to classical networks;
  - Architectural modifications – rectified linear activations (*ReLUs*);
  - Availability of massive datasets: *ImageNet* + efficient *GPU* computing hardware;
- Deeper architectures capture better invariant properties of the data comparing to shallow networks;
- Ability to generalize from a small number of training examples.

1. Introduction

1.2 Properties of Deep Neural Networks

- Design of Deep Neural Networks: Approximate arbitrary functions of the input

  Neural Networks with a single hidden layer and sigmoid activations => universal function approximators

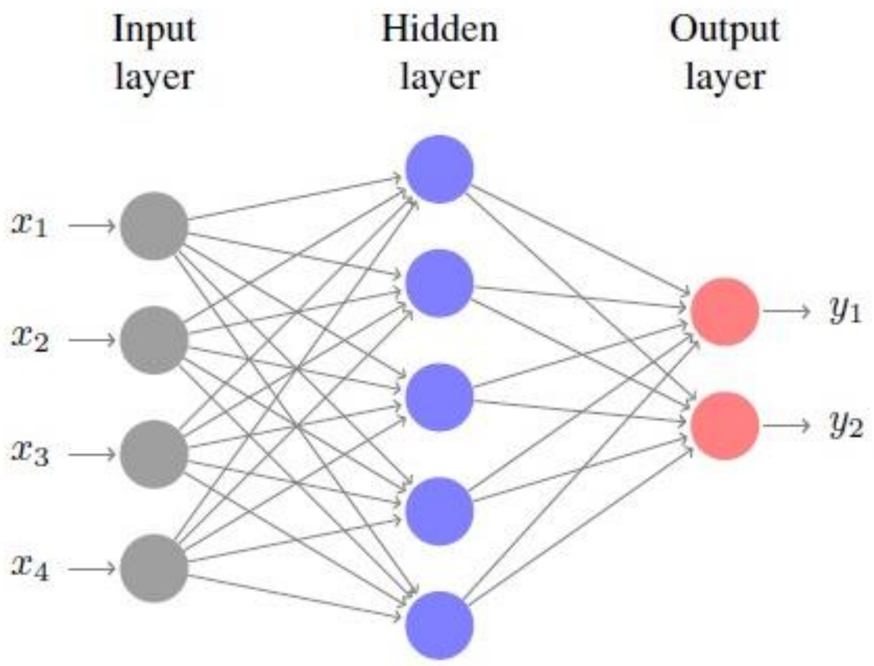
- Statistical Learning Theory: Number of training examples needed to achieve good generalization grows polynomially with the size of the network, but deep networks are trained with fewer data than the number of parameters $N \ll D$

- Another key property of a network architecture:
  Ability to produce ‘‘good representation of the data’’

  → Representation: any function of the input data that is useful for a task and a optimal one can be quantified by information-theoretic and complexity

1. Introduction

Figure 1: Illustration of a neural network with 4 inputs, 5 hidden layers and 2 outputs



Source: SOATTO, S; GIRYES, R; BRUNA, J; VIDAL, R. Mathematics of Deep Learning. [1]

1. Introduction

1.3 Approach for techniques and Mathematical Methods

- For complex data tasks, data may be corrupted by '*nuisances*' -→ One goal it to make the representation invariant to '*nuisances*'

- In general, **optimal representations** for a task can be defined as sufficient statistics which are minimal and **invariant to nuisance variability to future tests**.

- Optimization Properties:

  Classical approach to training neural network → Minimize the loss using backpropagation (Gradient Descent Method, Stochastic, applied to neural networks).

  ↳ SGD (Stochastic Gradient Descent) approximates the gradient for massive datasets.

2. Mathematical Approach

2.1 How we can use Mathematics in Deep Learning?

- Linear Algebra, Probability/Statistics and Optimization are the mathematical pillars of Machine Learning.
- **Goal**: Constructing a function which can classify the training data correctly, so it can generalize to unseen test data [2]
- The **inputs** of the Function F are **vectors** and **matrices**.
- For the situation of **identifying handwritten digits**, each input sample will be an image - a matrix of pixels. So, each one of the images will be **classified as a number from 0 to 9** [2].
- Assign weights to different pixels in the image to create the function.
- However, the key challenge is to **choose weights so that the function assigns the correct output**.

2. Mathematical Approach

2.2 Building a Function

- The **inputs** are the samples v and the **outputs** are the computed classification $w = F(v)$ [2];
- Simplest linear functions would be the **linear**: $w = Av$, the **entries of the matrix** $\boldsymbol{A}$ are the **weights** to be learned;
- It is also common to encounter the **bias vector** $\boldsymbol{b}$, so as the function may be defined: $F(v) = Av + b \quad (Affine)$;
- Since **linearity is very limiting requirement**, other functions were used to **establish non-linearity**: **sigmoidal functions** with $S - shaped\ graphs \ \rightarrow A\big(S(Bv)\big)$;
- After, it was verified that curved logistic functions S could be replaced by the **ramp function** $\boldsymbol{ReLU(x) = \max(0, x)}$**;**
- Functions of deep learning have the form $F(v) = L(R\left(L\left(R(\dots.(Lv))\right)\right)$

$$\rightarrow Composition\ of\ Affine\ functions\ Lv = Av + b$$
$$with\ non - linear\ functions\ R\ \rightarrow act\ on\ each\ component\ of\ the\ vector\ Lv$$

- The **matrices** $\boldsymbol{A}$ and the **bias vector** $\boldsymbol{b}$ are the **weights in the learning function**.

2. Mathematical Approach

2.2 Building a Function

- $F(x, v) \rightarrow depends\ on\ the\ input\ v\ and\ the\ weights\ x$
- The outputs $v_1 = ReLU(A_1(v) + b)$ from the first step produce the first hidden layer in the neural net.
- Beginning: input layer v
- Ending: output layer $w = F(v)$
- Affine part: $L_k(v_{k-1}) = A_k v_{k-1} + b_k$ of each step uses the computed weight $A_k\ and\ b_k$

2.3 Results and Loss Function

- Choose weights $A_k\ and\ b_k$ to minimize the total loss over all the training examples: the total loss the sum of each individual loss.
  - The loss function for least squares has the form:$||F_{(v)} - true\ output||^2$

2. Mathematical Approach

- **2.4 Optimization**: The goal is to minimize a Function $F(x_1, \ldots\ldots, x_n), where\ Derivate = zero\ at\ the\ minimum\ point\ x'$:

- So we have $n\ equations\ \frac{\partial F}{\partial x_i} = 0,\ for\ n\ unknows\ x'_1, \ldots, x'_n$

- There are conditions the **vector $\boldsymbol{x}$** must satisfy: These **constraints** could be **equations $\boldsymbol{Ax} = \boldsymbol{b}$**, $x \geq 0$. The constraints enter in the equation through Lagrange Multipliers $\lambda_1, \ldots .\lambda_m$.

- Expression **argmin**: $argmin\ F(x) = value(s)\ of\ x\ where\ F\ reaches\ its\ minimum$

- Important equations:

One Function F	$F(x + \Delta x) \approx F(x) + \Delta x \frac{dF}{dx}(x) + \left(\frac{1}{2}\right)(\Delta x)^2 \frac{d^2F}{dx^2}(x)$	(1)
One Variable x		

- The Function will be convex, its slope increase and its graph bends upward, when the second derivative of $F(x)\ is\ positive$: $\frac{d^2F}{dx^2} > 0$

2. Mathematical Approach

One Function $$F(x + \Delta x) \approx F(x) + (\Delta x)^T \nabla F + \left(\frac{1}{2}\right)(\Delta x)^T H(\Delta x) \quad (2)$$

Variables x_1 to x_n

- Second derivate matrix H is positive definite,

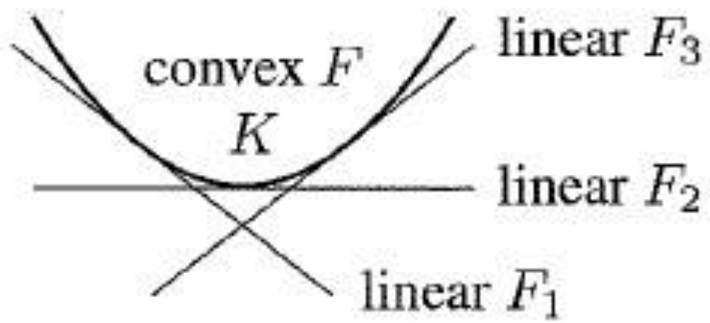
  ↳ F is a $strictly$ $convex$ $function$: **it is placed above its tangents**

- **2.5 Definition of convexity**

- A convex function F has a minimum at x' if $f = \nabla F(x') = 0$;
- Looking at all points $px + (1-p)y$ between x and y, so the graph of F stays on or goes below a straight line graph.
- F is convex: $F(px + (1-p)y) \leq pF(x) + (1-p)F(y)$ for $0 < p < 1$ (3)
- Then the graph of F goes below the chord that connects the point $P_1 = (x, F(x))$ to $P_2 = (y, F(y))$ and stays above its tangent lines.

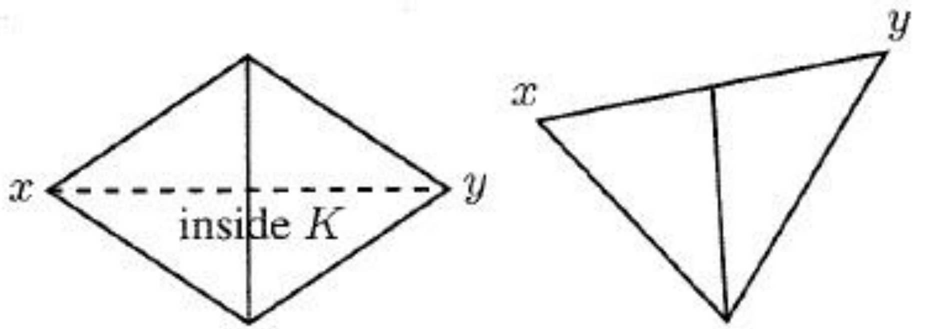
2. Mathematical Approach

Figure 2: A Convex Function F is the maximum of its all tangent functions



Source: STRANG, G. Linear Algebra and Learning from Data, Massachusetts Institute of Technology [2]

Figure 3: Two convex sets in $\mathbb{R}^2$



Source: STRANG, G. Linear Algebra and Learning from Data, Massachusetts Institute of Technology [2]

2. Mathematical Approach

2.5 Definition of convexity

- The maximum of 2 or more linear functions is rarely linear, but the **maximum $\boldsymbol{F(x)}$ of 2 or more convex $\boldsymbol{F_i(x)}$ is always convex**.
- For any $z = px + (1-p)y, between\ x\ and\ y$, each function F_i:

$$F_i(z) \leq pF_i(x) + (1-p)F_i(y) \leq pF(x) + (1-p)F(y) \qquad (4)$$
$$which\ is\ true\ for\ each\ i$$

- Then $F(z) = \max F_i(x) \leq pF(x) + (1-p)F(y)$
- An ordinary Function $f(x)$ is convex if $\frac{d^2F}{dx^2} \geq 0$. The extension of $n\ variables$ demands for the $n\ x\ n\ matrix\ \mathcal{H}(x)$ of second derivates.
- If F(x)is a smoth function, so there is a good test for convexity:

$$F(x_1, \ldots \ldots x_n)\ is\ convex\ if\ and\ only\ if\ its\ second\ derivative\ matrix\ \mathcal{H}(x)\ is\ positive\ semidefinite\ at\ all\ x.$$
$$The\ function\ F\ is\ strictly\ convex\ if\ \mathcal{H}(x) is\ positive\ definite\ at\ all\ x$$

$$\mathcal{H}(x) = \begin{bmatrix} \frac{\partial^2 F}{\partial x_1^2} & \frac{\partial^2 F}{\partial x_1 x_2} & \ldots\ldots.. \\ \frac{\partial^2 F}{\partial x_2 x_1} & \frac{\partial^2 F}{\partial x_2^2} & \ldots\ldots. \\ & \ldots\ldots.. & \end{bmatrix} \qquad (5)$$

3. Mathematical Notation

❑ 3.1 Symbols and Sets for DNNs

- Deep Networks are a hierarchical model where each layer applies a *linear transformation* + *nonlinearity* to **the preceding layer**
- Let $X \in \mathbb{R}^{N\,x\,D}$: the input data, where each row of X is D-dimensional data point and N is the number of training examples
- Let $W^k \in \mathbb{R}^{d_{k-1}\,x\,d_k}$: matrix representing a linear transformation applied to the output of layer $k-1$
- $X_{k-1} \in \mathbb{R}^{N\,x\,d_{k-1}}$: the output of layer $k-1$
- $X_{k-1}W^k \in \mathbb{R}^{N\,x\,d_k}$: d_k- dimensional representation at layer k
- Each column of W^k represent a convolution with some filter (CNNs)

3. Mathematical Notation

3.1 Symbols and Sets for DNNs

- Let $\varphi_k: \mathbb{R} \to \mathbb{R}$ to be a nonlinear activation function
  - $\varphi_k = \tanh(x)$
  - $\varphi_k = (1 + e^{-x})^{-1}$
  - $\varphi_k = \max\{0, x\}$
- This nonlinearity is applied to each entry of the $X_{k-1}W^k$ to generate the k_{th} layer of the neural network as:

$$X_k = \varphi_k(X_{k-1}W^k)$$

- The output of the network is given by:

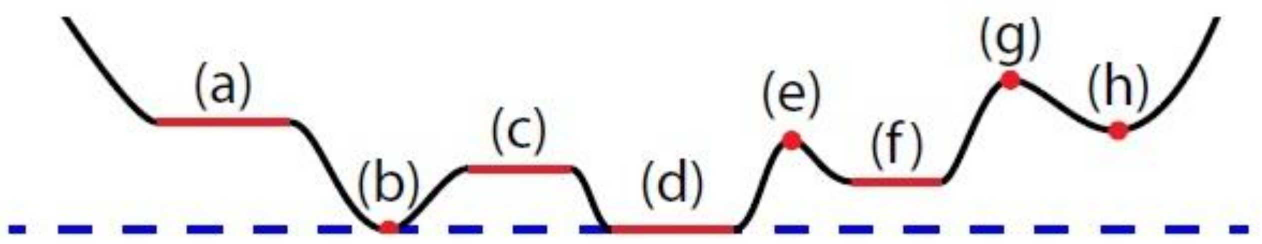
$$\Phi\left(X, W^1, \ldots, W^k\right) = \varphi_k(\varphi_{k-1}(\ldots \varphi_2(\varphi_1(XW^1)W^2)\ldots W^{k-1})W^k)$$

⟶ Φ is matrix with dimensions $N \; x \; C$, $\mathrm{C} = d_k$ is the dimension of the output of the network, which is the number of classes for a classification task

3. Mathematical Notation

Figure 4: Example of critical points of non-convex function

(a,c): Plateaus; (b,d): Global Minima; (e,g): Local Maxima; (f,h): Local Minima



Source: SOATTO, S; GIRYES, R; BRUNA, J; VIDAL, R. Mathematics of Deep Learning. [1]

3. Mathematical Notation

3.2 Global Optimality

- Learning the parameters $W = \{W^k\}_{k=1}^{K}$ of deep network from N training examples (X, Y).
- A row of $X \in \mathbb{R}^{NxD}$ represents a data point in $\mathbb{R}^D$;
- A row of $Y \in \{0,1\}^{NxC}$ represents the membership of each data point to one out of C classes:
- Y_{jc}= 1 if j_{th} row of X belongs to class $c \in \{1, \ldots., C\}$ or $Y_{jc} = 0$ in the opposite case;
- The problem of learning the network weights W could be stated as follows:

$$\min \; l\left(Y, \Phi\left(X, W^1, \ldots, W^k\right)\right) + \lambda\Theta\left(W^1, \ldots\ldots W^k\right), \{W^k\}_{k=1}^{K} \qquad (6)$$

- $l(Y, \Phi)$ is the **loss function** that measures the **agreement** between the **predicted output Φ** and the **true output Y;**
- Θ is a **regularization function to prevent overfitting**, $\Theta = \sum_{k=1}^{K} \left|\left|W^k\right|\right|_F^2$
- $\lambda > 0$ is a balancing parameter.

4. Results and Discussion

4.1 Non-convexity in neural network training

- The previous optimization problem is **non-convex** due to the map $\Phi(X, W)$, which is a non-convex function of W, due to the product of W^k variables and the nonlinearities ψ_k.

- For non-convex problems, the set of critical points includes not only the global minima but also local minima, local maxima, saddle points and saddle plateau.

$$\rightarrow model\ formulation + implementation\ details(\ inicialization\ of\ the\ model\ and\ optimization\ algorithm)$$

- Dealing with non-convexity in deep learning requires initialization of the networks weights at random and update these weights with local descent , check if the training error is decreasing fast and if not, choose another inicialization.

4. Results and Discussion

4.2 Optimality for DNNS with single hidden layer

- If the size of the network is large enough and non-linearity is the $ReLU$, many weights are $zero$, occurs a phenomenon known as $dead\ neurons$, improving the classification performance.

- Later work also discovered that for neural networks with a single hidden layer, if the number of neurons in the hidden layer is not fixed but fit to the data, so the process of training a globally optimal neural network is analogous to selecting a finite number of hidden units from a potentially infinite dimensional space of all possible hidden units.

- The optimization problem is stated as follows, which the **output** is reckoned as the **weighted sum** of the **selected hidden units**:

$$\min l(Y, \sum_i h_i(X)w_i) + \lambda||w||_1 \qquad (7)$$

- $\boldsymbol{h_i(X)}$ represents **one of all possible hidden unit activation** due to the training data X from an **infinite dimensional** space $h_i(X) \in \mathcal{H}$

4. Results and Discussion

4.2 Optimality for DNNS with single hidden layer

- The primary difficult is how to select the appropriate hidden linear units because $\mathcal{H}$ is an infinite dimensional space.
- However from gradient boosting, is possible to show that it can be globally optimized by sequentially adding hidden units to the network until one can no longer find a hidden unit whose addition will decrease the objective function.

4.3 Global Optimality for positively homogeneous networks

- Some authors proposed by considering certain assumptions, the **critical point of a high-dimensional optimization** is more likely a **saddle point rather than a local minimizer**.
- **Avoiding saddle points is the main challenge** in high-dimensional non-convex optimization.
- Besides, under some assumptions on the distribution of the training data and network parameters, other authors show that with the increasing number of hidden units in a network, the distribution of local minima becomes concentrated in a small intervals of objective function values near the global optimum.
- Generally the conditions for non-convex optimization problems have an approach considering all critical points to be either global minimizers or saddle points/plateaus.

4. Results and Discussion

4.4 Geometric Stability

- Mathematically characterize its approach: define the class of regression and classification tasks for which they are predesigned to perform well.
- For **Computer Vision** tasks, CNNs provide a fundamental inductive idea of the **origin of successful deep learning vision models**.

4.4.1 Framework to understand: Geometric Stability

- Let $\Omega = [0,1]^d \sqsubset \mathbb{R}^d$ be a compact d −dimensional Euclidean Domain on which square-integrable functions $X \in L^2(\Omega)$ are defined : Images can be thought as functions on the unit square $\Omega = [0,1]^2$

Supervised learning task, an unknown function $f: L^2(\Omega) \rightarrow \Upsilon$ on a training set:

$$\{X_i \in L^2(\Omega), Y_i = f(X_i)\}_{i \in I} \qquad (8)$$

- Target Space Υ is discrete in a standard classification setup, where $C = |\Upsilon|$

4. Results and Discussion

4.4.2 Geometric Properties

- In computer vision and speech analysis tasks, the unknown function f satisfies the following crucial assumptions:

1. **Stationarity**: Considering a Translation Operator

$$T_v X(u) = X(u - v), u, v \in \Omega \qquad (9)$$

- Acts on functions $X \in L^2(\Omega)$. It can be supposed the function to be invariant with respect to translations. In the object classification tasks, $f(T_v X) = f(X)$, for any $X \in L^2(\Omega)$ and $v \in \Omega$
- Or it can also be assumed equivariant: $f(T_v X) = T_v f(X)$, well-defined when the output of the model is a space in which translations can act upon (problems of object localization).

2. **Local deformations and scale separations**

- $\mathcal{L}_\tau$ is deformation where $\tau\colon \Omega \to \Omega$ is smooth vector field acts on $L^2(\Omega)$ as $\mathcal{L}_\tau X(u) = X(u - \tau(u)) \qquad (10)$

4. Results and Discussion

- Deformations can model local translations, changes in viewpoint and rotations.

- **Tasks in computer vision are not only translation invariant, but also stable with respect to local deformations**. Therefore in tasks which are translation invariant:

$$|f(L_\tau X) - f(X)| \approx \left||\nabla_\tau|\right| \qquad (11)$$

- For all X and τ, $\left||\nabla_\tau|\right|$ measures the **smoothness of a deformation field.** So the quantity predicted does not change much if the input image is slightly deformed.

- For tasks which are translation equivariant:

$$|f(L_\tau X) - \mathcal{L}_\tau f(X)| \approx \left||\nabla_\tau|\right| \qquad (12)$$

- That is a strong property since the space of local deformations has high dimensionality, order of $\mathbb{R}^D$, when discretize images with D pixels, opposed to $d - dimensional$ translation group, where $d = 2$ dimensions for images.

5. Bibliographic References

- **[1]** SOATTO, S; GIRYES, R; BRUNA, J; VIDAL, R. **Mathematics of Deep Learning**, 13 December 2017.
- **[2]** STRANG, G. **Linear Algebra and Learning from Data**, Massachusetts Institute of Technology, Wellesley-Cambridge Press.