UniTs - University of Trieste

Faculty of Scientific and Data Intensive Computing

Department of mathematics informatics and geosciences

Deep Learning

Lecturer:
Prof. Alessio Ansuini

Author:
Andrea Spinelli

March 18, 2025

This document is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike (CC BY-NC-SA) license. You may share and adapt this material, provided you give appropriate credit, do not use it for commercial purposes, and distribute your contributions under the same license.

github.com/Spina02 andreaspinelli2002@gmail.com

Preface

As a student of Scientific and Data Intensive Computing, I've created these notes while attending the **Deep Learning** course.

The prerequisites of the course are basic knowledge of:

- Linear Algebra (eigenvalue problems, SVD, etc.)
- Mathematical Analysis (multivariate differential calculus, etc.)
- Probability (chain rule, Bayes theorem, etc.)
- Machine Learning (logistic regression, PCA, etc.)
- Programming (Python, Linux Shell, etc.)

While these notes were primarily created for my personal study, they may serve as a valuable resource for fellow students and professionals interested in Deep Learning.

Draft

Contents

Draft

1 Introduction

1.1 What is Deep Learning?

Definition: *Deep Learning*

*"**Deep Learning** is constructing networks of parametrized functional modules and training them from examples using gradient-based optimization."*

~ Yann LeCun

In other words, Deep Learning is a collection of tools to build complex modular differentiable functions. These tools are devoid of meaning, it is pointless to discuss what DL can or cannot do. What gives meaning to it is how it is trained and how the data is fed to it.

Deep learning models usually have an architecture that is composed of multiple layers of functions. These functions are called **modules** or **layers**. Each layer is a function that takes an input and produces an output. The output of one layer is the input of the next layer. The output of the last layer is the output of the model.

Practical Applications of Deep Learning

Deep Learning has revolutionized numerous fields by achieving unprecedented performance on complex tasks. In **computer vision**, convolutional neural networks can recognize objects, detect faces, segment images, and even generate realistic images. **Protein structure prediction** has seen remarkable advances with models like AlphaFold, which can accurately predict 3D protein structures from amino acid sequences, fundamentally changing molecular biology and drug discovery. **Speech recognition and synthesis** systems powered by deep learning can transcribe spoken language with near-human accuracy and generate natural-sounding speech, enabling voice assistants and accessibility tools. Other applications include natural language processing (powering chatbots and translation systems), recommendation systems, anomaly detection in cybersecurity, weather forecasting, and autonomous driving. The versatility of deep learning comes from its ability to learn meaningful representations directly from data, reducing the need for manual feature engineering while achieving superior performance across diverse domains.

1.2 Family of linear functions

Let's consider a simple modell

$$y = \Phi_0 + \Phi_1 x$$

This model is a linear function of x with parameters Φ_0 and Φ_1. The model can be represented as a line in the $x - y$ plane. The parameters Φ_0 and Φ_1 determine the slope and the intercept of the line.

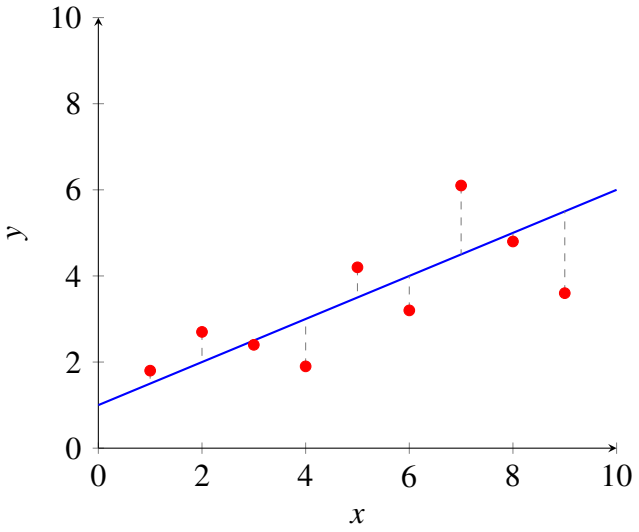


Figure 1.1: Linear model with scatter points and error segments

Loss Function

The loss function for this model is the mean squared error (MSE) between the predicted values and the actual values:

$$L = \frac{1}{N}\sum_{i=1}^{N}(\underbrace{\Phi_0 + \Phi_1 x_i}_{=P_i} - y_i)^2$$

where N is the number of data points, x_i is the i-th input, y_i is the i-th target, and P_i is the predicted value for the i-th data point.

Optimization

The goal of optimization is to find the values of Φ_0 and Φ_1 that minimize the loss function. This is done by computing the gradient of the loss function with respect to the parameters and updating the parameters in the opposite direction of the gradient. The update rule for the parameters is given by:

Let's calculate the gradient of the losso function:

$$\nabla_{\{\Phi\}} L = \left[\frac{\partial L}{\partial \Phi_0}, \frac{\partial L}{\partial \Phi_1}\right]$$

Then we can update the parameters as follows:

$$\begin{cases} \Phi_0 \leftarrow \Phi_0 - \lambda \dfrac{\partial L}{\partial \Phi_0} \\ \Phi_1 \leftarrow \Phi_1 - \lambda \dfrac{\partial L}{\partial \Phi_1} \end{cases} \quad \Rightarrow \quad \Phi^{new} = \Phi^{old} - \lambda \nabla_{\{\Phi\}} L$$

where λ is the **learning rate**, a hyperparameter that controls the size of the parameter updates.

This is only a step in the optimization process. The optimization algorithm iteratively updates the parameters until the loss converges to a minimum. But when shell we stop?

$$|L^{new} - L^{old}| < \varepsilon$$

where ε is a small positive number that determines the convergence threshold.

Exercises

Let's talk further about the loss function:

$$L = \frac{1}{N}\sum_{i=1}^{N}(\underbrace{\Phi_0 + \Phi_1 x_i}_{=P_i} - y_i)^2$$

The aim is to minimize the loss function. In this case the loss is a paraboloid function of Φ_0 and Φ_1, so it has a single minimum.

Questions

1. Calculate the gradient of the Loss function
2. Find the minimum of the loss function
3. Show that the gradient of L is orthogonal to the level lines
4. Estimate the computational complexity of the exact solution of the linear regression problem in the general case (take into account the number of data samples and the number of dimensions/features used)

$$\Phi = (X^\top X)^{-1} X^\top y$$

Solutions

1. Calculate the gradient of the Loss function
2. Find the minimum of the loss function
3. Show that the gradient of L is orthogonal to the level lines
4. Estimate the computational complexity of the exact solution of the linear regression problem in the general case (take into account the number of data samples and the number of dimensions/features used)

   $$\Phi = (X^\top X)^{-1} X^\top y$$

   Let's consider a $X_{N\times D}$ matrix and let's computate the complexity of the single operations:
   - $(x^\top X)$ is a $D \times D$ matrix, so the complexity is $O(ND^2)$
   - $(x^\top X)^{-1}$ requires $O(D^3)$ operations
   - $(x^\top y)$ is a $D \times 1$ matrix, so the complexity is $O(ND)$

   The total complexity is $O(ND^2 + D^3 + \cancel{ND}) = O(ND^2 + D^3)$

2 Shallow Neural Networks

2.1 Activation Functions

2.1.1 Rectified Linear Unit (ReLU)

The ReLU activation function is defined as:

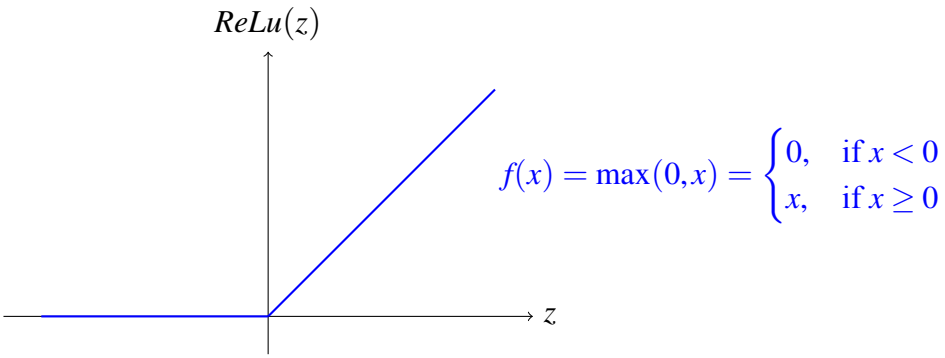


Figure 2.1: Rectified Linear Unit (ReLU) activation function

2.1.2 Elements of the family F

$$y = \phi_0 + \phi_1 a[\theta_{10} + \theta_{11}x] + \phi_2 a[\theta_{20} + \theta_{21}x] + \phi_3 a[\theta_{30} + \theta_{31}x]$$

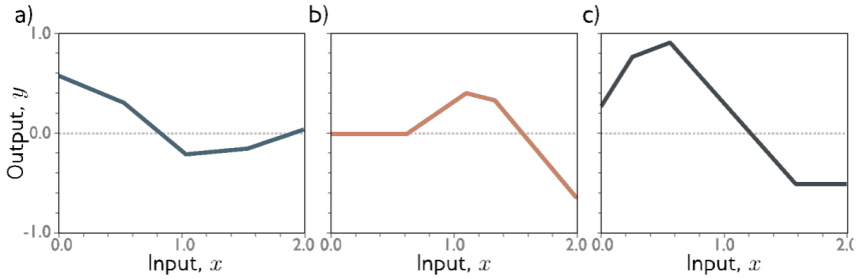
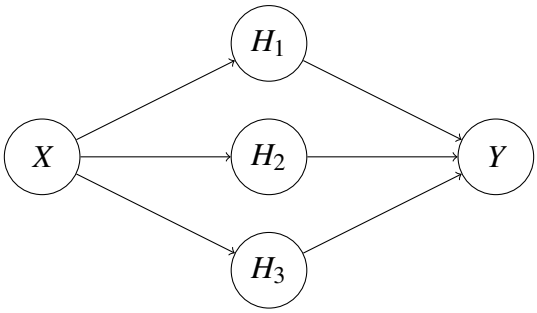


Figure 2.2: Family of functions F

$$\begin{cases} z_1 = \phi(a_1) = \phi_0 + \phi_1 a[\theta_{10} + \theta_{11}x] \\ z_2 = \phi(a_2) = \phi_0 + \phi_2 a[\theta_{20} + \theta_{21}x] \\ z_3 = \phi(a_3) = \phi_0 + \phi_3 a[\theta_{30} + \theta_{31}x] \end{cases}$$



$$y = W^{(2)}\phi(W^{(1)}x + b^{(1)}) + b^{(2)}$$

$$W^* = W^{(2)}W^{(1)} \qquad \text{rank}(W^*) = \min$$

…

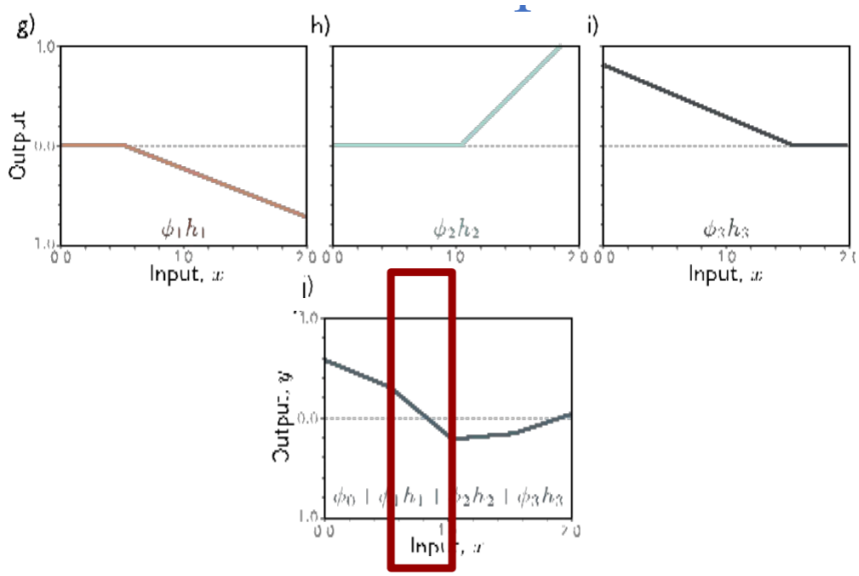
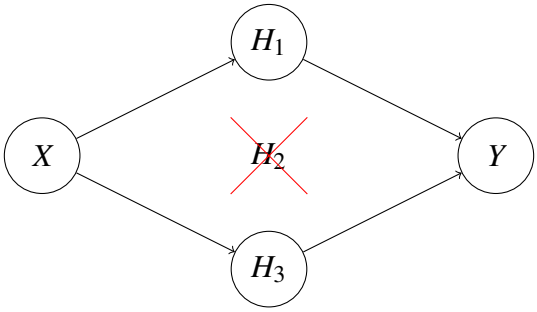


Figure 2.3: Flow computation

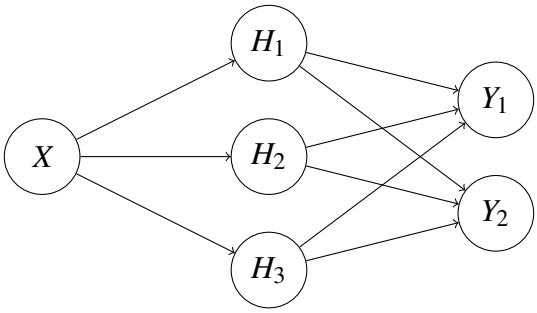
In the hilighted section, the contribute of the 2nd neuron of the hidden layer is zero, so the output is only influenced by the 1st and 3rd neurons of the hidden layer.

This is a consequence of the ReLU activation function, which deactivates the second node in that interval of the input space:



Multivariate Output

For multivariate output, the situation is similar to the previous case. In this case, the output layer has two neurons, Y_1 and Y_2:



We obtain the output as:

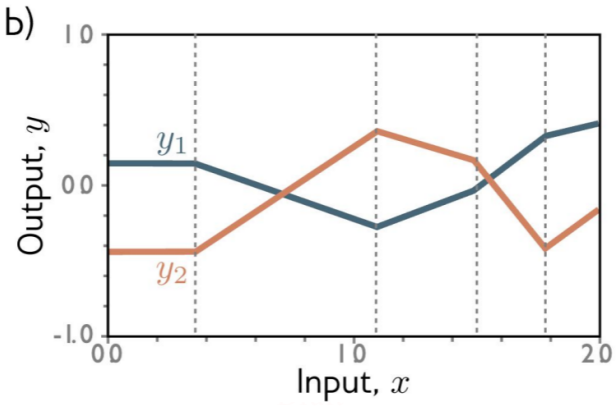
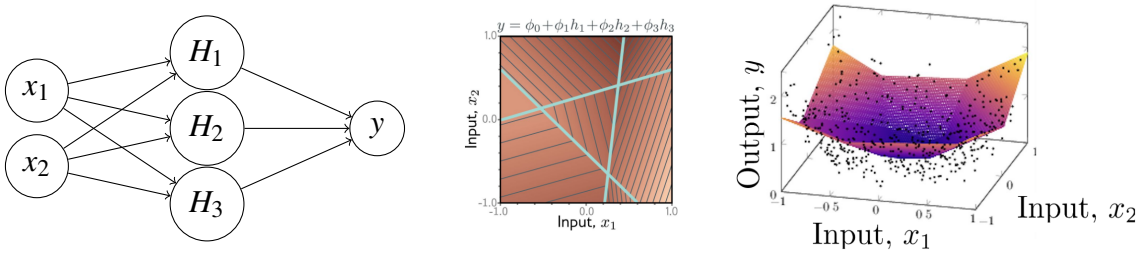


Figure 2.4: Flow computation for multivariate output

Multivariate Input

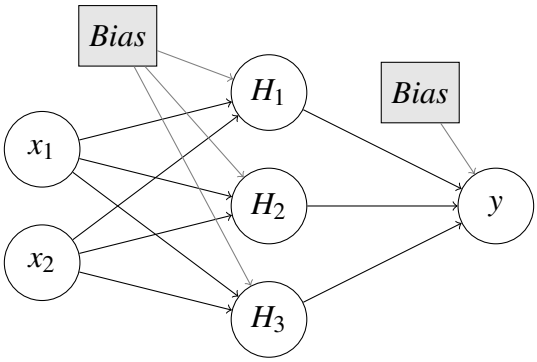


In this case we have 13 parameters:

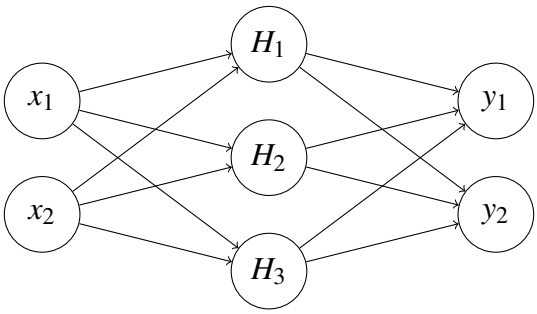
- We have 3 parameters from X_1 and X_2 to connect each node of the hidden layer (2 are the weights and 1 is the bias), for a total of 9 parameters.
- We have 4 parameters from the hidden layer to the output layer (3 weights and the bias term).

The general formula to count the number of parameters is:

#



General Case

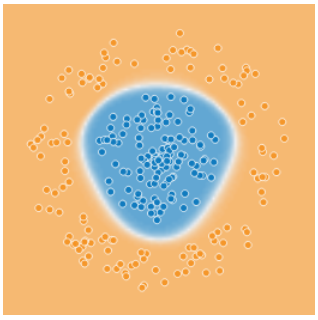


Draft

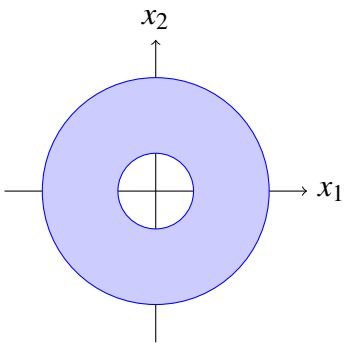
3 Lecture 18/03/2025

Polar Coordinates

Let's consider data disposed as in figure:

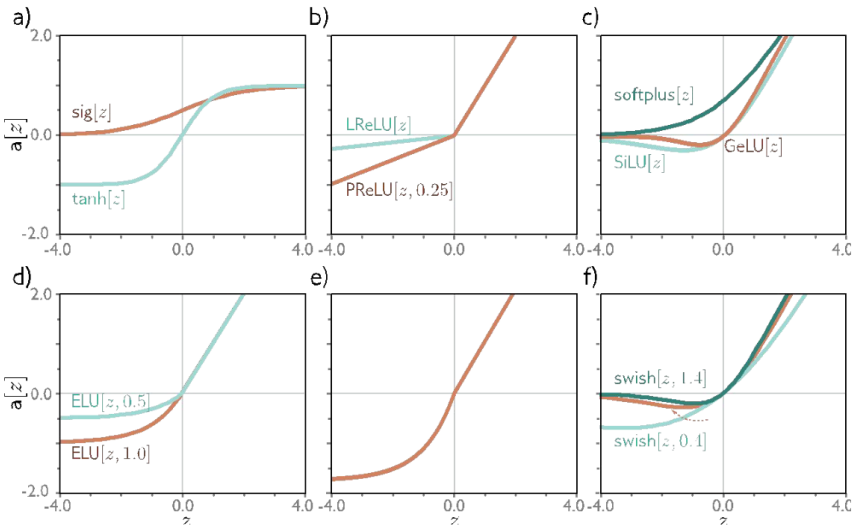


We can convert the data from Cartesian to Polar coordinates as follows:



$$\begin{cases} x_1 \\ x_2 \end{cases} \rightarrow \begin{cases} r = \sqrt{x_1^2 + x_2^2} \\ \theta = arctg\dfrac{x_2}{x_1} \end{cases}$$

Other Non-Linearities



3.1 Universal Approximation Theorem

Let σ be any continuous discriminatory function.

$$\text{Then finite sums of the form} \quad G(x) = \sum_{j=1}^{N} \alpha_j \sigma(y_i^\top x + \theta_j) \quad \text{are dense in} C(I_n).$$

In other words, given any $f \in C(I_n)$ and $\varepsilon > 0$, there is a sum, $G(x)$, of the above form, for wich:

$$|G(x) - f(x)| < \varepsilon \qquad \forall x \in I_n$$

—

Any function $f(x)$ on $\mathbb{R}^d$ with some smoothness conditions can be approximated by a single hidden layer sigmoidal neural network $f_n(x)$ with n hidden units such that:

$$\int_{B_r} (f(x) - f_n(x))^2 \mu(dx) \leq \frac{c}{n}$$

where μ is a probability measure on ball $B_r = \{x : |x| < r\}$, c is a constant independent of n and r.

Observation: ***Feature Learning Advantage***

A result of the Universal Approximation Theorem is that no linear combinain of n fixed basis functions yells integrated square error smaller than order:

$$\left(\frac{1}{n}\right)^{\frac{2}{d}}$$

3.2 Number of Linear Regions

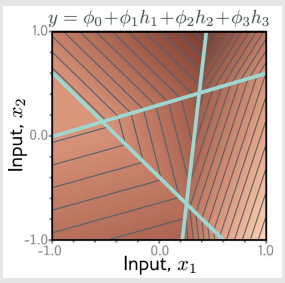
Be D_i the size of the input layer, D the size of the hidden layer, Then, given $D_i \leq D$.
The number of linear regions N is given by:

$$N \leq \sum_{j=0}^{D_i} \binom{D}{j}$$

Example:

Let's consider $D_i = 2$ and $D = 3$. The number of linear regions is given by:

$$N \leq \binom{3}{0} + \binom{3}{1} + \binom{3}{2} = 1 + 3 + 3 = 7$$



Fixed Functions

$$y = \sum_{w_i} \phi(W_{i_{D_i}} x_{D_i} + b_i) \sim \{B_i(x)\}$$

Polynomial case

If the dimension is $DIM = 1$ we have something of the form:

$$y = w_0 + w_1 x + w_2 x^2 + \cdots + w_M x^M$$

If the dimension is $DIM = D$, we have:

$$y = w_o + \sum_{i=1}^{D} w_i x_i + \sum_{i,j=1}^{D} w_{ij} x_i x_j + \cdots + \sum_{i_1,\ldots,i_D=1}^{D} w_{i_1,\ldots,i_D} x_{i_1} \ldots x_{i_D}$$

The consequence is that the number of parameters grows exponentially with the dimension.

Curse of dimensionality

$$B_i(x) = \begin{cases} 0 & \text{if } x \text{ does not face in the block } b_i \\ majority & \text{if the class of } x \text{ in the block } A \end{cases}$$

Draft