Introduction to Deep Learning

Mnacho Echenim

Grenoble INP-Ensimag

2022-2023

Notes

About this class

- Deep learning
  - Very dynamic field of research
    - Increased computational power
    - Increased data availability
  - Different kinds of neural networks, depending on applications
    - Convolutional networks for image processing
    - Recursive neural networks, large language models for natural language processing
    - Generative adversarial network for physical simulations and image processing
  - But not many impressive results obtained by using a neural network as a black box, with no knowledge on how they work or on the data
    - Other ML techniques can outperform neural networks, depending on the problem at hand
- Goal of this class
  - Present the fundamentals of deep learning by implementing an efficient Multilayer Perceptron (MLP) from scratch
  - Use this MLP to solve a regression problem (**supervised learning**)
  - **Nb:** you may feel lost at the beginning of the hands-on labs but everything will fall into place with time

Notes

Main outline

- Origins of neural networks and deep learning
- Main properties of neural networks
- Learning techniques
  - Basis (gradient descent, overfitting, . . .)
  - Improvements (gradient optimization, regularization, . . .)
- Application to a pricing problem
  - Question: can a neural network be used as a proxy for a costly function?

Notes

Timetable

1. Introduction, multilayer perceptrons, forward propagation
   - Hands-on: forward propagation
2. Gradient descents, backpropagation
   - Hands-on: backpropagation for stochastic gradient descent
3. Mini-batch gradient descent, gradient descent optimization
   - Hands-on: mini-batch gradient descent
4. Hands-on: mini-batch, Momentum
5. Regularization
   - Hands-on: L2 regularization
6. Feature preprocessing, batch normalization
   - Pricing project
7. Energy considerations
   - Pricing project
8. Pricing project

Notes

Some resources

- Goodfellow, Bengio, Courville. Deep Learning: https://www.deeplearningbook.org/
- Aggarwal. Neural Networks and Deep Learning
- Nielsen. Neural Networks and Deep Learning: http://neuralnetworksanddeeplearning.com
- Lectures by Mallat at Collège de France: https://www.college-de-france.fr/site/stephane-mallat/
- Lectures by Le Cun at Collège de France: https://www.college-de-france.fr/site/yann-lecun/
- Stanford School of Engineering channel: https://www.youtube.com/user/stanfordeng

Notes

Some landmarks

1943 W. McCulloch, W. Pitts: *A Logical Calculus of the Ideas Immanent in Nervous Activity*

- Formal model of a neuron
- Weighted inputs, binary outputs

1948 N. Wiener: *Cybernetics: Or Control and Communication in the Animal and the Machine*

- “Self-regulating mechanisms”
- Notion of feedback

1950 A. Turing: The Turing Test

- Proposal: build a program simulating a child’s mind, then educate it

1951 M. Minsky and D. Edmonds: Stochastic Neural Analog Reinforcement Calculator (SNARC)

- First neural network machine

Notes

A classification problem

1958. F. Rosenblatt: *The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain*

- Input: $\{(\chi_{(i)}, \rho_{(i)}) \mid i = 1, \dots, N\}$, where $\chi_{(i)} \in \mathbb{R}^d$ and $\rho_{(i)} \in \{+1, -1\}$
- Goal: construct a classifier that correctly labels the training set
- Perceptron: outputs $\operatorname{sign}(\omega^{\mathrm{T}}\chi + \beta)$
- Problem: find ω, β such that $\forall i = 1, \dots, N$, $\operatorname{sign}(\omega^{\mathrm{T}}\chi_{(i)} + \beta) = \rho_{(i)}$
- Reformulation: find ω, β such that $\forall i = 1, \dots, N$, $\rho_{(i)} \cdot (\omega^{\mathrm{T}}\chi_{(i)} + \beta) > 0$

Notes

Parameter update algorithm

Notations

We define:

- $\theta \stackrel{\text{def}}{=} (\beta, \omega_1, \dots, \omega_d)^{\mathrm{T}}$ and $\overline{\chi} \stackrel{\text{def}}{=} (1, \chi_1, \dots, \chi_d)^{\mathrm{T}}$

Goal: find θ such that $\forall i = 1, \dots, N$, $\rho_{(i)} \cdot (\theta^{\mathrm{T}}\overline{\chi_{(i)}}) > 0$

Input: $\{(\chi_{(i)}, \rho_{(i)}) \mid i = 1, \dots, N\}$

$\theta \leftarrow (0, \dots, 0)^{\mathrm{T}}$;
while $\exists j \in \{1, \dots N\}$ *such that* $\rho_{(j)} \cdot \operatorname{sign}(\theta^{\mathrm{T}}\overline{\chi_{(j)}}) < 0$ **do**
  $\theta \leftarrow \theta + \rho_{(j)}\overline{\chi_{(j)}}$
end
return θ

Notes

Convergence

Theorem

If there exists a separating hyperplane for the samples, then the algorithm terminates and the resulting perceptron correctly classifies all samples

- Separating hyperplane: $\exists \overline{\omega^*}$ such that for all $i = 1, \ldots, N$, $\rho_{(i)} \cdot (\overline{\omega^*}^{\mathrm{T}} \overline{\chi_{(i)}}) > 0$
  - Note that $\overline{\omega^*} \neq \mathbf{0}$
- Proof principle: determine an upper-bound on the number k of iterations in the **while** loop
- We let:
  - $\delta \stackrel{\text{def}}{=} \min_i \left[\rho_{(i)} \cdot (\overline{\omega^*}^{\mathrm{T}} \overline{\chi_{(i)}})\right]$; note that $\delta > 0$
  - $R \stackrel{\text{def}}{=} \max_i \left\|\overline{\chi_{(i)}}\right\|$; note that $R > 0$
- We have (See, e.g., `http://www.cs.columbia.edu/~mcollins/courses/6998-2012/notes/perc.converge.pdf`):

$$k \leq \frac{R^2 \left\|\overline{\omega^*}\right\|^2}{\delta^2}$$

Notes

What if the samples admit no separating hyperplane?

Minsky & Papert (1969): *Perceptrons: an introduction to computational geometry*

- The XOR function does not admit a separating hyperplane
- The perceptron is **incapable** of learning this function

Notes

Summary on the perceptron

- Simple learning scheme
- Convergence result when there exists a separating hyperplane
  - It is possible to evaluate the convergence speed
- Minsky & Papert: there are simple functions that cannot be learned
  - This negative result slowed research on neural networks for a long time
- Question: what happens if we generalize the perceptron by
  - Interconnecting several neurons together
  - Allowing activation functions other than the sign function?

Notes

Definitions

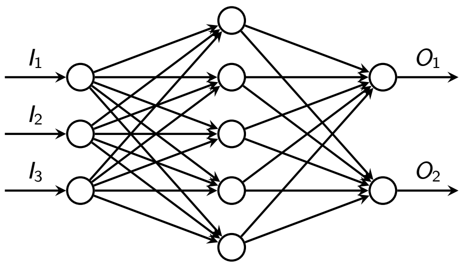
Definition

A multilayer perceptron (MLP, or feedforward neural network) is a set of interconnected neurons that forms a Directed Acyclic Graph (DAG)

- The connections between neurons are **weighted**
- Neurons with no incoming connection from another neuron are **input neurons**. The set of input neurons forms the **input layer**
- Neurons with no outgoing connection to another neuron are **output neurons**. The set of output neurons forms the **output layer**
- All other neurons are **hidden neurons**. Hidden neurons are organized in **hidden layers**
- Computations are performed in the hidden and output layers.

Notes

Fully connected multilayer perceptron (3 layers)



Definition

A fully connected MLP is an MLP such that all non-input neurons are connected to all the neurons of the previous layer

In what follows, all the MLPs we consider will be fully connected

Notes

Notations and assumptions

Given a fully connected MLP with L layers:

- The neurons of layer i are denoted by $\nu_1^i, \ldots, \nu_{n_i}^i$ (layer 0 denotes the input layer)
- The weight connecting ν_j^{i-1} to ν_k^i is denoted by $\omega_{j,k}^i$
- The bias of ν_k^i is denoted by β_k^i
- The sample transmitted to the input layer is denoted by $\alpha^0 = (\alpha_1^0, \ldots, \alpha_{n_0}^0)^{\mathrm{T}}$
- The **net input** of ν_k^i is $\zeta_k^i \stackrel{\text{def}}{=} \left(\sum_{j=1}^{n_{i-1}} \omega_{j,k}^i \alpha_j^{i-1}\right) + \beta_k^i$
- The **activation** of ν_k^i is denoted by α_k^i
- The activation function of ν_k^i is Φ, so that $\alpha_k^i = \Phi(\zeta_k^i)$
  - Unless stated otherwise (e.g., **Dropout**), we assume it is the same for all neurons

Assumption

In what follows, unless specified otherwise, we will assume that the MLPs we consider have a single output node

Notes

Matrix representation

- The weights at layer i are stored in a matrix $\Omega^i \in \mathbb{R}^{n_{i-1} \times n_i}$, where $(\Omega^i)_{j,k} = \omega^i_{j,k}$
- The biases at layer i are stored in a vector $\beta^i \stackrel{\text{def}}{=} (\beta^i_1, \ldots, \beta^i_{n_i})^{\mathrm{T}}$
- The net inputs at layer i are stored in a vector $\zeta^i \stackrel{\text{def}}{=} (\zeta^i_1, \ldots, \zeta^i_{n_i})^{\mathrm{T}}$
- The activation function is extended to vectors: if $u = (u_1, \ldots, u_n)^{\mathrm{T}}$, then $\Phi(u) = (\Phi(u_1), \ldots, \Phi(u_n))^{\mathrm{T}}$
- The activations of layer i are stored in a vector $\alpha^i \stackrel{\text{def}}{=} (\alpha^i_1, \ldots, \alpha^i_{n_i})^{\mathrm{T}}$

Notes

Forward propagation algorithms

Input: A network with L layers
Input: $\alpha^0 \stackrel{\text{def}}{=} (\alpha^0_1, \ldots, \alpha^0_{n_0})^{\mathrm{T}}$

for $i \leftarrow 1$ **to** L **do**
  for $k \leftarrow 1$ **to** n_i **do**
    $\zeta^i_k \leftarrow \left(\sum_{j=1}^{n_{i-1}} \omega^i_{j,k} \alpha^{i-1}_j\right) + \beta^i_k$;
    $\alpha^i_k \leftarrow \Phi(\zeta^i_k)$;
  end
end

Algorithm 2: Forward propagation, basic version

Notes

Identity activation function

What happens if the activation function is the identity?

- $\alpha^i = \zeta^i = \left[\Omega^i\right]^{\mathrm{T}} \alpha^{i-1} + \beta^i$

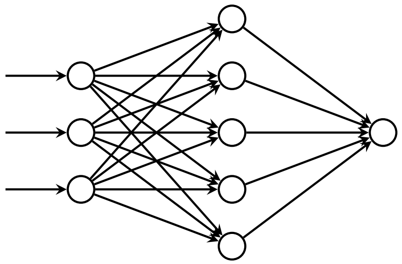
Notes

Linear activation function

What happens if the activation function is linear: $\Phi : z \mapsto \mu z + \lambda$?

Notes

A specific MLP



- Input layer of size d
- Single hidden layer of size K
- Single output neuron
- Activation function Φ only on hidden layer (identity on output)
- No bias on output layer

Notes

A class of functions

- We define $\Lambda \stackrel{\text{def}}{=} \left\{\langle K, \Phi, \Omega, \omega', \beta\rangle \,\middle|\, K \in \mathbb{N},\ \Phi : \mathbb{R} \to \mathbb{R},\ \Omega \in \mathbb{R}^{d\times K},\ \beta, \omega' \in \mathbb{R}^K\right\}$
- For $\lambda \in \Lambda$, we define

$$\begin{array}{rcl} f_\lambda :\ \mathbb{R}^d & \to & \mathbb{R} \\ x & \mapsto & [\omega']^{\mathrm{T}} \cdot \Phi\left(\Omega^{\mathrm{T}} x + \beta\right) = \sum_{k=1}^{K} \omega'_k \Phi([\Omega_k]^{\mathrm{T}} x + \beta_k) \end{array}$$

- We consider the set of functions $\mathcal{F}(\Lambda) \stackrel{\text{def}}{=} \{f_\lambda \mid \lambda \in \Lambda\}$

Question

What functions can be approximated by an element from $\mathcal{F}(\Lambda)$?

Notes

The Universal Approximation Theorem

Theorem (see, e.g., Allan Pinkus, 1999)

Let $\mathcal{C}\left(\mathbb{R}^d\right)$ denote the set of continuous functions on $\mathbb{R}^d$ and consider the set $\mathcal{M}(\Phi) \stackrel{def}{=} \operatorname{span}\left(\left\{x \mapsto \Phi(\omega^{\mathrm{T}} x + \beta) \,\middle|\, \omega \in \mathbb{R}^d, \beta \in \mathbb{R}\right\}\right)$. This set is dense in $\mathcal{C}\left(\mathbb{R}^d\right)$, in the topology of uniform convergence on compact sets if and only if Φ is nonpolynomial

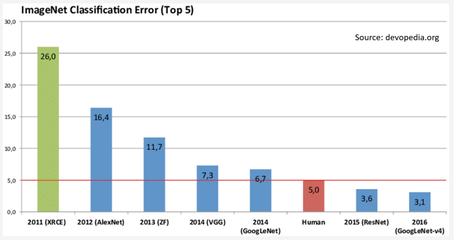
In other words: if $f \in \mathcal{C}\left(\mathbb{R}^d\right)$ and $\mathcal{D} \subseteq \mathbb{R}^d$ is a compact set, then for all $\varepsilon > 0$, there exists $\widetilde{f} \in \mathcal{M}(\Phi)$ such that for all $x \in \mathcal{D}$, $|f(x) - \widetilde{f}(x)| \leq \varepsilon$

- For a given $\varepsilon > 0$, there exists $K_\varepsilon \in \mathbb{N}$ such that an MLP with a single hidden layer of size K_ε can approximate f with a precision of ε
- What is wrong with polynomials?

Notes

Issues with the Universal Approximation Theorem

- The size of the hidden layer can be extremely large
  - In practice, this result does not seem very useful
- What happens if we increase the number of hidden layers and bound their sizes?
  - For a long time, not so clear there was anything to gain
  - Empirically yes: impressive results using deep networks



  - More recent theoretical results (Eldan, Shamir. The power of Depth for Feedforward Neural Networks, 2016)
  - For the time being, it is still not possible to formally explain why things work so well

Notes