Introduction to Deep learning with R for dummies by dummies

S. Donnet

Paris, March 2019

Introduction : supervised learning

Example: Digit Recognition

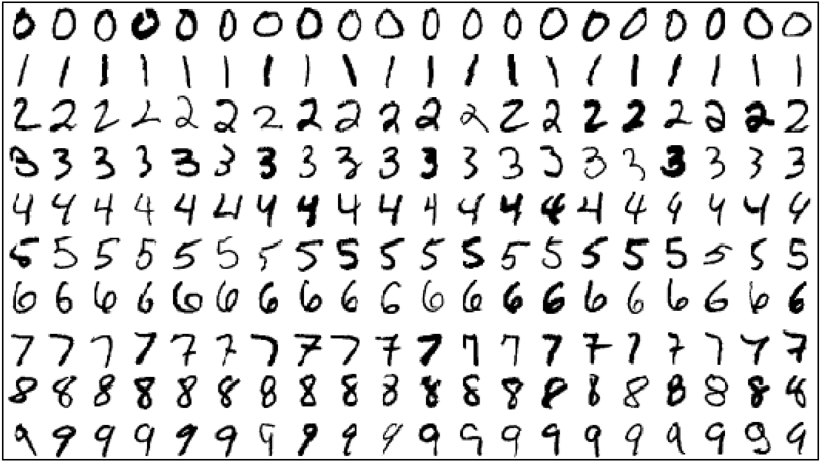


Figure 1: Recognition of handwritten digits

Example: Digit Recognition

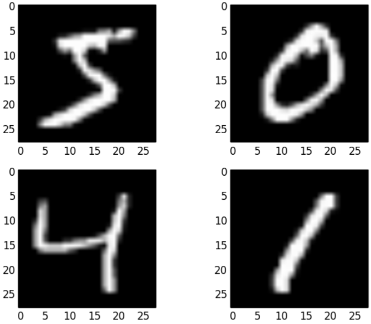


Figure 2: Recognition of handwritten digits

- Each image i of digit is of size $28 \times 28 = 784$ pixels
- Each pixel j is associated with a gray level $x_i^j \in \{0, \dots, 255\}$

Example: Digit Recognition

- The gray levels are stored in a vector $\mathbf{x}_i = (x_i^r)_{r=1\ldots p}$, $p = 784$
- $y_i \in \{0, \ldots, 9\}$: image label
  - y_i : observed / known for a learning sample
  - y must be predicted for a new image $\mathbf{x}$

Formally

- We consider n objects (images, texts, sound ...), described by p characteristics.
- For each object i, these characteristics are stored in a vector $\mathbf{x}_i = (x_i^1, \ldots, x_i^p)$ of $\mathbb{R}^p$.
- An output variable y_i is each object i is assigned
  - If $y_i \in \mathbb{R}^d$: we talk about regression
  - If $y_i \in E$ with E set, we talk about - discriminating if $E = \{0, 1\}$; - classification if $E = \{0, \ldots, 9\}$ for example - shape recognition if $E = \{\text{dog}, \text{groundhog}, ...\}$
- **Goal**: predict the output y for a new set of characteristics $\mathbf{x}$
- **How do we do it?**: learn (on a set of learning data = training) a prediction rule or classification and provide this rule to apply it to $\mathbf{x}$

Other examples

- Face recognition on pictures $E = \{$ family members $\}$
- Recognition of the political edge by speach analysis

What's the difference between estimation and learning?

- Statistical tradition / estimation :
  - Concept of model is central with an explanatory purpose
  - Seeking to approach reality, propose a model possibly based on a physical theory, economic,
  - Interpretation of the role of each explanatory variable in the process is important.
- Learning: the objective is essentially the prediction,
  - best model is not necessarily the one that best fits the true model.
  - theoretical framework is different and error control requires another approach: choices based on prediction quality criteria
  - Interpretability is less important

So what about the good old (generalized) linear model?

$$y_i \sim \mathcal{F} \quad \text{with} \quad \phi(\mathbb{E}[y_i]) = \mathbf{x}^T \beta$$

- If the dimensions of the problem (n, p) are reasonable
- If model assumptions (linearity) and distributions are verified
- THEN: statistical modeling techniques issued from the general linear model are optimal (maximum likelihood)
- We will not do better, especially in the case of small samples

But. . .

- As soon as the distribution hypotheses are not verified,
- As soon as the supposed relations between the variables or with the variable of interest are not linear
- As soon as the data volume is important (big data),

We will consider other methods that will over-rate rudimentary statistical models.

Deep learning

Deep learning: introduction

- Definition (attempt): set of learning methods that attempt to model data with complex architectures combining various non-linear transformations

$$\mathbf{x} \mapsto f(\mathbf{x}, \theta) \text{ such that } y \simeq f(\mathbf{x}, \theta)$$

- The basic building blocks of Deep Learning are **neural networks**.
- These bricks are combined to form **deep neural networks**.

Application areas

These techniques have led to significant progress in the following areas:

- image and sound processing: facial recognition, automatic speech recognition (transformation of a voice into written text),
- computer vision: to imitate the human vision (machine seeing several objects at once),
- automatic processing of natural language,
- text classification (eg spam detection).

Infinite amount of potential applications

Different types of architectures for neural networks

- *Multilayer Perceptrons*: the oldest and simplest
- *Convolutional neural networks*: very efficient for image processing
- *Recurrent neural networks*, useful for sequential data (texts or time series)
- All are based on deep layers of layers
- Requires intelligent optimization algorithms (stochastic in general), careful initialization and a smart choice of structure.
- Impressive results but few theoretical justifications for the moment

Artificial neuron

- *A neuron* is a non-linear application in its parameters that associates an output $f(\mathbf{x})$ to an input vector $\mathbf{x}$:
- More precisely, the j-th neuron f_j is written

$$f_j(\mathbf{x}) = \phi(< w_j, \mathbf{x} > + b_j)$$

where

  - $< w_j, \mathbf{x} >= \sum_{r=1}^{p} w_j^r x^r$
  - The quantities $w_j = (w_j^1, \ldots, w_j^p)$ are the weights of the input variables $(x^1, \ldots, x^p)$
  - $b_j$is the bias of neuron j.
  - ϕ is the activation function

Basic model involving one neuron

We explain the output variable y by

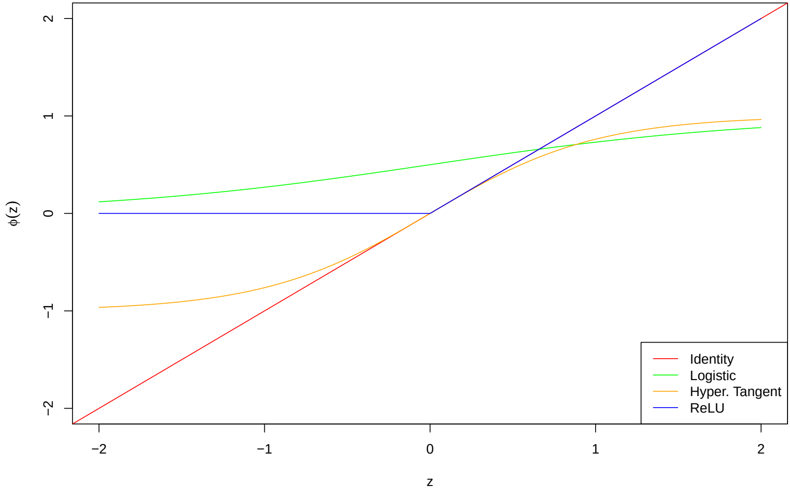
$$y \simeq f(\mathbf{x}, \theta) = \phi(< w, \mathbf{x} > + b)$$

- If $\phi(z) = z$ and we only have one neurone $\Rightarrow$ classical linear function

Choice of the activation function ϕ

- If $Y \in \{0,1\}$ and we want to predict $P(Y=1|x)$: logistic function
$$\phi(z) = \frac{1}{1+e^{-z}} \in [0,1]$$
- *Generalization*: If $Y \in E$ (E finite space of cardinal K) and we want to predict $P(Y=e|x)$: *softmax*
$$\phi(z_1,\ldots,z_K) = \left(\frac{e^{z_j}}{\sum_{j=1\ldots K} e^{z_j}}\right)_{j=1\ldots K}$$
- Hyperbolic tangent function : $\phi = \tanh : \mathbb{R} \mapsto [-1,1]$
- Threshold function : $\phi(z) = 1_{z>s} \in \{0,1\}$
- Rectified linear unit (ReLU) $\phi(z) = max(z,0)$
  - OK to predict positive values. Continuous but non differentiable

Plots



Various differentiability properties : important when we will have to optimize the w et b

Neural networks or multilayer perceptrons

- Structure composed of different hidden layers of neurons whose outputs serve as inputs to the neurons of the next layer
- Activation functions are the same in the different layers, only the last is different (adapted to the objective: classification or regression)

Example

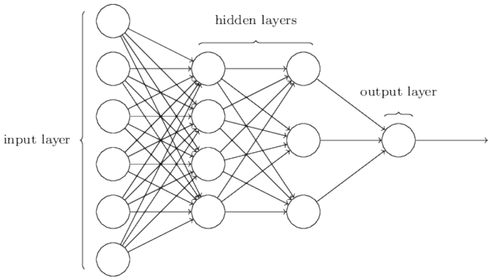


Figure 3: Example of neural network

- Input layer : as many nodes as variables x : p
- Hidden layer : number of neurons, to be set but the user (here 4 then 3)
- Output layer : 1 node $= y$ for regression or binary classif and number of modalities for general classification

Neural network : formally

Let J_ℓ be the number of neurons in layer ℓ

- Layer 0 : $h^0(\mathbf{x}) = \mathbf{x} \in \mathbb{R}^p$
- For hidden layers $\ell = 1 \dots L$:
  - We create J_ℓ neurons : for every $j = 1 \dots J_\ell$:

$$\begin{aligned} a_j^{(\ell)}(\mathbf{x}) &= b_j^{(\ell)} + \sum_{m=1}^{J_{\ell-1}} W_{jm}^{(\ell)} h_m^{(\ell-1)}(\mathbf{x}) \\ &= b_j^{(\ell)} + < W_j^{(\ell)}, h^{(\ell-1)}(\mathbf{x}) > \end{aligned}$$

```
$$h_j^{(\ell)}(\mathbf{x}) = \phi(a_j^{(\ell)}(\mathbb
```

- With vectors and matrices :

$$a^{(\ell)}(\mathbf{x}) = b^{(\ell)} + W^{(\ell)} h^{(\ell-1)}(\mathbf{x}) \in \mathbb{R}^{J_\ell}$$

$$h^{(\ell)}(\mathbf{x}) = \phi(a^{(\ell)}(\mathbf{x}))$$

where $W^{(\ell)}$ is matrix of size $J_\ell \times J_{\ell-1}$

Neural network: formally

- For the last layer $\ell = L + 1$:

$$a^{(L+1)}(\mathbf{x}) = b^{(L+1)} + W^{(L+1)} h^{(L)}(\mathbf{x}) \in \mathbb{R}^J$$
$$h^{(L+1)}(\mathbf{x}) = \psi(a^{(L+1)}(\mathbf{x}))$$

Neural network: finally

- $W^{(\ell)}$ is a weight matrix with J_ℓ rows and $J_{\ell-1}$ columns.
- $W^{(L+1)}$ is a weight matrix with 1 row and J_L colums $y \in \mathbb{R}$
- $$\mathbf{x} \mapsto f(\mathbf{x}, \theta) = \psi(a^{(L+1)}(\mathbf{x}))$$

  - If we are in a regression context $\psi(z) = z$,
  - If we are in a binary classification context ψ is the sigmoid function (prediction in $[0, 1]$).
  - If we are in a multiclass classification framework : $\psi = softmax$

Neural Network:

- Basic architecture since each layer depends on the previous layer and not on the neurons of the same layer ($\Rightarrow$ recurrent neural networks)
- Parameters to tune or fit: - number of layers - number of neurons in each layer - hidden layer activation functions (ϕ) - Choice of the output activation function (ψ) driven by the dataset

Recurrent neural networks

- The output of a neuron can be the input of a neuron of the same layer.

Theoretical result

- Hornik (1991) proved that any smooth bounded function from $\mathbb{R}^p$ to $\mathbb{R}$ can be approximated by a one layer neural network with a finite number of neurons with the same activation function ϕ and $\psi = id$.
- Interesting result from a theoretical point of view.
- In practice :
  - Required number of neurons can be very large.
  - Strength of deep leargning derives from the number of hidden layers

Parameters estimation

A quantity to minimize / maximize : loss function

- Parameters to estimate : $\theta =$ weights $W^{(\ell)}$ ans bias $b_j^{(\ell)}$
- Classically : estimation by maximizing the (log)-likelihood
- Loss function: $= -\log$ likelihood
  - In the regression framework : $Y \sim \mathcal{N}(f(\mathbf{x},\theta), I)$

$$\ell(f(\mathbf{x},\theta), Y) = \|Y - f(\mathbf{x},\theta)\|^2$$

  - For the binary classification $\{0,1\}$: $Y \sim \mathcal{B}(1, f(\mathbf{x},\theta))$

$$\ell(f(\mathbf{x},\theta), Y) = -Y\log(f(\mathbf{x},\theta)) - (1-Y)\log(1-f(\mathbf{x},\theta))$$

    (cross-entropy)
  - For the multiclass classification

$$\ell(f(\mathbf{x},\theta), Y) = -\sum_{e\in E}\mathbf{1}_{Y=e}\log p_\theta(Y=e|\mathbf{x})$$

Loss function: remark

- Ideally, we would like to minimize the classification error but it is not a differentiable function
- So we resort to the "cross-entropy" which can be differentiated

Penalized empirical risk

Risk

$$\mathbb{E}_Y[\ell(f(\mathbf{x},\theta), Y)]$$

replaced by the empirical risk $\mathbb{E}_Y[\ell(f(\mathbf{x},\theta), Y)]$

- For a training sample $(\mathbf{x}_i, Y_i)_{i=1\ldots n}$

$$L_n(\theta) = \frac{1}{n}\sum_{i=1}^{n} \ell(f(\mathbf{x}_i,\theta), Y_i)$$

- Penalization :

$$L_n(\theta) = \frac{1}{n}\sum_{i=1}^{n} \ell(f(\mathbf{x}_i,\theta), Y_i) + \lambda\Omega(\theta)$$

with, for instance, $\Omega(\theta) = \sum_{\ell,i,j}(W_{ij}^{(\ell)})^2$ or $\Omega(\theta) = \sum_{\ell,i,j}|W_{ij}^{(\ell)}|$ to avoid overfitting.

Minimization by Stochastic gradient descent.

Algorithm (by Rumelhart et al (1988))

- Choose an initial value of parameters θ and a learning rate η
- Repeat until a minimum is reached:
  - Split randomy the training set into N_B *batches* of size m $(n = m \times N_B)$
  - for each batch B set:

$$\theta := \theta - \eta \frac{1}{m} \sum_{i \in B} \nabla_\theta \left\{ \ell(f(\mathbf{x}_i, \theta), Y_i) + \lambda \Omega(\theta) \right\}$$

Remarks:

- Each iteration is called an *epoch*.
- The number of epochs is a parameter to tune
- Difficulty comes from the computation of the gradient

Calculus of the gradient for the regression

- $Y \in \mathbb{R}$.
- $R_i = \ell(f(\mathbf{x}_i, \theta), Y_i) = (Y_i - f(\mathbf{x}_i, \theta))^2$
- For any activation function ϕ (hidden layers) and ψ

Partial derivatives of R_i with respect to the weights of the last layer

- Derivatives of $R_i = (Y_i - f(\mathbf{x}_i, \theta))^2 = \left(Y_i - h^{(L+1)}(\mathbf{x}_i)\right)^2$ with respect to $(W_j^{(L+1)})_{j=1\dots J_L}$
- $a^{(L+1)}(\mathbf{x}) = b^{(L+1)} + W^{(L+1)} h^{(L)}(\mathbf{x}) \in \mathbb{R}^J$
- $$\begin{aligned} f(\mathbf{x}, \theta) &= h^{(L+1)}(\mathbf{x}) \\ &= \psi(a^{(L+1)}(\mathbf{x})) \\ &= \psi\left(b^{(L+1)} + \sum_{j=1}^{J_L} W_j^{(L+1)} h_j^{(L)}(\mathbf{x})\right) \end{aligned}$$
- $$\frac{\partial R_i}{\partial W_j^{(L+1)}} = -2\left(Y_i - f(\mathbf{x}_i, \theta)\right)\psi'\left(a^{(L+1)}(\mathbf{x}_i)\right) h_j^{(L)}(\mathbf{x}_i)$$

Partial derivatives of R_i with respect to the weights of the layer $L-1$

- Derivatives of $R_i = \left(Y_i - h^{(L+1)}(\mathbf{x}_i)\right)^2$ with respect to $(W_{jm}^{(L)})_{j=1\ldots J_L, m=1\ldots J_{L-1}}$
- $$\frac{\partial R_i}{\partial W_{jm}^{(L)}} = -2\left(Y_i - f(\mathbf{x}_i,\theta)\right)\psi'\left(a^{(L+1)}(\mathbf{x}_i)\right)\frac{\partial}{\partial W_{jm}^{(L)}} a^{(L+1)}(\mathbf{x}_i)$$

Partial derivatives of R_i with respect to the weights of the layer $L-2$

$$
\begin{aligned}
a^{(L+1)}(\mathbf{x}) &= b^{(L+1)} + \sum_{j=1}^{J_L} W_j^{(L+1)} h_j^{(L)}(\mathbf{x}) \\
&= b^{(L+1)} + \sum_{j=1}^{J_L} W_j^{(L+1)} \phi\left(b_j^{(L)} + \sum_{m=1}^{J_{L-1}} W_{jm}^{(L)} h_m^{(L-1)}(\mathbf{x}) \right)
\end{aligned}
$$

$$
\begin{aligned}
\frac{\partial}{\partial W_{jm}^{(L)}} a^{(L+1)}(\mathbf{x}_i) &= W_j^{(L+1)} \phi'\left(b_j^{(L)} + \sum_{m=1}^{J_{L-1}} W_{jm}^{(L)} h_m^{(L-1)}(\mathbf{x}_i) \right) \\
&\quad \times h_m^{(L-1)}(\mathbf{x}_i) \\
&= W_j^{(L+1)} \phi'(a_j^L(\mathbf{x}_i)) h_m^{(L-1)}(\mathbf{x}_i)
\end{aligned}
$$

Forward-Backward algorithm (at each iteration)

After some light effort, recurrence formula

- Given the current parameters
  - **Forward step** : From layer 1 to layer $L+1$, compute the $a_j^\ell(\mathbf{x}_i), \phi(a_j^\ell(\mathbf{x}_i))$
  - **Backward step** : From layer $L+1$ to layer 1, compute the partial derivatives (recurrence formula update)

Tuning the algorithm

- η: learning rate of the gradient descent
  - if η too small, really slow convergence with possibly reaching of a local minimum
  - if η too large, maybe oscilliation around an optimum without stabilisation
  - Adaptive choice of η (decreasing η)
- Batch calculation reduces the number of quantities to be stored in the forward / backward

Obviously

Many improved versions of the maximisation algorithm (momentum correction, Nesterov accelerated gradient, etc. . .)

Automatic differentiation

Success of the neural network comes from automatic differentiation, i.e. automatisation of the previously described forward-backward procedure to compute the derivatives : `Tensorflow`

More complexe neural networks

Convolutional neural network (CNN)

- LeNet by LeCun et al., 1998
- When we transform an image into a vector : loss of spatial coherence of the image (shapes, . . .)
- CNN revolutionized Image Analysis (LeCun)
- Composed of different convolution layers, pooling layers and fully connected layers.

In one picture

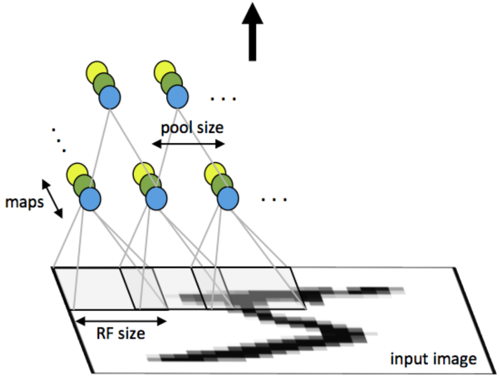


Figure 4: Architecture of a convolutive neural network (Stanford credit)

Convolution layer

- Image $\mathcal{I}(u, v, c)$: each pixelwith (u, v) is described by C color levels (channels), par exemple RGB (red, green, blue) $\Rightarrow$: array of size (M, N, C)
- Each neuron relies on a linear combination of the signals of a small region of the image:

$$K_{u,v} * \mathcal{I}(c) = \sum_{n=-k}^{k} \sum_{m=-k}^{k} K_l(n, m, c)\mathcal{I}\left(u + m, v + n, c\right)$$

$$h_{u,v} = \phi(K_{u,v} * \mathcal{I}(c) + b_{u,v})$$

- We obtain a new neuron by moving this window - The amount by which the filter shifts is the **stride** - if we don't move a lot : redundancy of information - if we move a lot: loss of information

Pooling layer (subsampling)

- Averaging or computing maximum on small areas
- ϕ can be applied before or after pooling
- Allows to reduce the dimension (number of variables to be treated) but also to make the network less sensitive to possible translations of the image

Finally

- After several layers and convolution / pooling, we apply one or more layers of *fully connected layers* (= network shown before)

Towards increasingly complex network architectures

- Networks now much more complex
- Can be processed with Graphical Processor Unit (GPU) cards
- Results of the *Large Scale Visual Recognition Challenge (ILSVRC)*

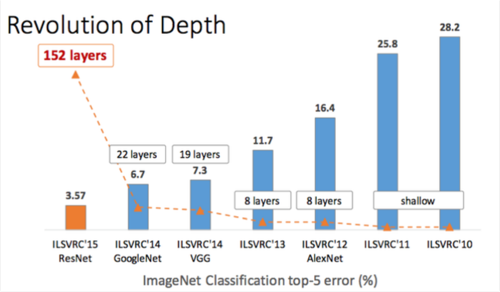
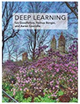


Figure 5: Error rate and depths of the network

Conclusion

Conclusion

- Ultra-fast vision of the definition of a deep neural network
- In practice: expertise is based on how to combine layers and different types of neurons (see practice here after)
- References: - Wikipedia - Wikistat course by Philippe Besse (Toulouse) - Reference book: Deep Learning by Yoshua Bengio




```
- Video https://www.college-de-france.fr/site/yann-lecu
- Results and conferences by Francis Bach
```