

Introduction to Deep Neural Networks and their application to Physics

Alberto Guillén and Luis Javier Herrera

University of Granada

14 September 2018

Contents

- ▶ Introduction to Machine Learning and Deep Learning
- ▶ Application Example
- ▶ Conclusions and Future work

Approximate Definition

- ▶ Machine learning
 - ▶ A wide variety of techniques, technologies and models to tackle problems like: Classification, Natural Language Processing, Regression, Games, etc
 - ▶ “What separates machine learning from optimization is that we want the generalization error , also called the test error, to be low aswell”
- ▶ Soft Computing:
it is a meta discipline, also known as Computational Intelligence, which includes metaheuristics and bioinspired systems as well

Types of problems

- ▶ According to the INPUT given to the algorithm
 - ▶ Unsupervised $X = [\vec{x}^i]$ with $i = 1 \dots m$ and $\vec{x}^i \in \mathbb{R}$ vs. Supervised learning (\vec{x}^i, y^i) with $i = 1 \dots m$ and $\vec{x}^i \in \mathbb{R}$
 - ▶ In supervised, regression $Y = [y_i] \in \mathbb{R}$ and classification $y_i \in label_1, label_2, \dots$ are the most common problems.

Type of learning

Online vs. Offline \rightarrow in On-line samples arrive through time meanwhile in Offline all are processed at the same time

Getting the "right" data set

When dealing with X we should keep in mind

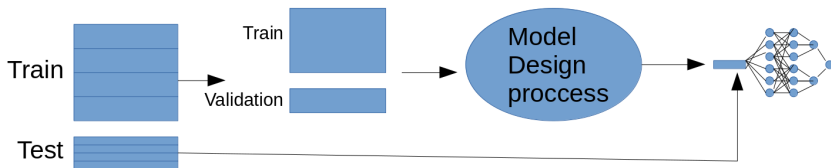
- ▶ Missing values?
 - ▶ Imputation
- ▶ Balanced?
 - ▶ Resampling, Boosting, Weighing...
- ▶ Noise or Outliers?

Regarding the variables

- ▶ Feature selection
- ▶ Feature extraction

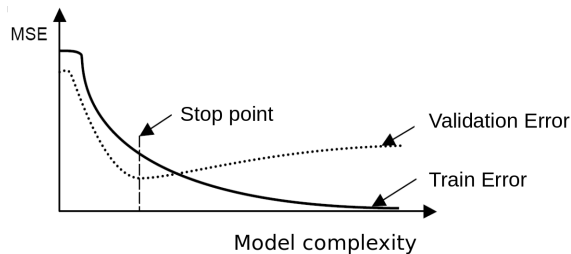
Getting the "right" data set (II)

Define Train, Test and Validation data sets



Why?

To avoid overfitting and select the best model from the different options



Model design

- ▶ Each model has its own hyperparameters → difficult (or impossible in some cases) to find out optimal configuration
- ▶ Use metaheuristics to perform the optimization
 - ▶ Genetic algorithms
 - ▶ Genetic programming
 - ▶ Ant Colony
 - ▶ Swarm Particle
 - ▶ Tabu search
 - ▶ Simulated annealing
 - ▶ ...

k -Nearest Neighbours

- ▶ Idea: inputs with similar values will have similar outputs

Using all data available $\{(\vec{x}_i, y_i)\}_{i=1}^D$ compute

$$\hat{F}(\vec{x}; k) = \frac{1}{k} \sum_{j=1}^k y_{NN_j(\vec{x})} \quad (1)$$

where $NN_j(\vec{x})$ is the j -th closest neighbour to \vec{x}

Weighted k -Nearest Neighbours

- ▶ The closer the neighbour is the higher should be the influence

$$F_{WkNN}(\vec{x}; k) = \frac{\sum_{j=1}^k w_j(\vec{x}) y_{NN_j(\vec{x})}}{\sum_{j=1}^k w_j(\vec{x})} \quad (2)$$

where $w_j(\vec{x}) = (1 - d_j^2(\vec{x})/d_{k+1}^2(\vec{x}))^2$ and $d_j(\vec{x})$ the distance of \vec{x} to the j -th closest neighbour.

Kernelised version

The distance can be replaced by a kernel obtaining a feature space like with kernel methods.

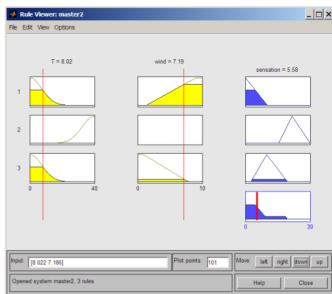
Fuzzy Models

They try to model the human way of thinking by using fuzzy sets and fuzzy models

IF T is "cold" AND Wind is "strong" THEN Sensation IS = "Very Cold"

IF T is "hot" THEN Sensation IS = "OK"

IF T is "cold" AND Wind is "soft" THEN Sensation IS = "Cold"



Fuzzy Models: Takagi-Sugeno-Kang (TSK)

The rule consequents are functions (instead of fuzzy sets) using the variables in the rule antecedent. Several types depending on the function computed:

▶ TSK-0:

IF $x^{(1)}$ is A AND $x^{(2)}$ is B then $z = R$

$$\hat{F}(\vec{x}) = \frac{\sum_{j=1}^{\#rules} \alpha_j \vec{x} R_j}{\sum_{j=1}^{\#rules} \alpha_j \vec{x}}$$

▶ TSK-1:

IF $x^{(1)}$ is A AND $x^{(2)}$ is B then $z = b + a\vec{x}$

$$\hat{F}(\vec{x}) = \frac{\sum_{j=1}^{\#rules} \alpha_j \vec{x} (b_j + a_j \vec{x})}{\sum_{j=1}^{\#rules} \alpha_j \vec{x}}$$

Support Vector Machines

The original idea was to find the hyperplane that separate the data in a linear way: $f(\vec{x}) = \text{sign}(\vec{w}^T \vec{x} + b)$.

As there could be infinite hyperplanes, SVM try to maximise the margin ($2/\|\vec{w}\|$), this is, to find out the support vectors. Thus, find \vec{w} and b such as $\forall(\vec{x}^{(i)}, y^{(i)}, i = 1 \dots D, y^{(i)}(\vec{w}^T \vec{x}^{(i)} + b) \geq 1$ minimising $\frac{1}{2} \|\vec{w}\|^2$.

Applying Lagrange multipliers we have:

$$L_p = \frac{1}{2} \|\vec{w}\|^2 - \sum_{i=1}^D \alpha_i [y^{(i)}(\vec{w}^T \vec{x}^{(i)} + b) - 1]$$

considering: 1) $\sum \alpha_i y^{(i)} = 0$ 2) $\alpha_i \geq 0 \forall \alpha_i$

Extensions

We can allow some misclassifications : Minimise

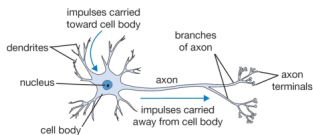
$\frac{1}{2} \|\vec{w}\|^2 + C \sum_{i=1}^D \xi_i$ We can use the "kernel trick" and map \vec{x} to a feature space where it might be (more) linearly separable.

Neural Networks

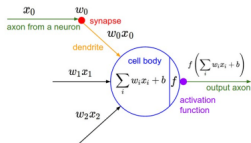
A (simplified) model of natural neural networks

Natural

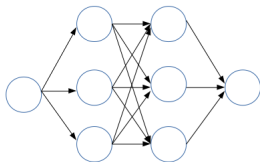
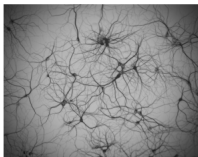
Synapsis



Artificial



Neural Network



1

¹Images: https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcSbb2Van-e2T24h3Z44c-HfUr4PXu-LcCNs3Gg20VdT3_aY1dR9ng
<http://cs231n.github.io/neural-networks-1/>

Neural Networks: A Neuron (Perceptron)

Given $(\vec{x}^{(i)}, y^{(i)})$ with $i = 1 \dots D$ and $\vec{x}^{(i)} \in \mathbb{R}$ define:

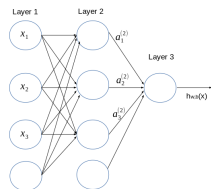
$$h_{\vec{w}, b}(\vec{x}) = f(\vec{w}^T \vec{x} + b) = f\left(\sum_{i=1}^d w_i x_i + b\right) \quad (3)$$

where f is called activation function.

Most common activation function for DL is ReLU (Rectified Linear Unit)

- ▶ Strong biological basis
- ▶ Computational advantages (solves the vanishing gradient if x_i becomes too large as its derivative is constant)

Neural Networks: A simple network



Parameters: (\vec{w}, b)

Notation:

- ▶ $w_{ij}^{(l)}$ = weight for connection between unit j in layer l and unit i in layer $(l + 1)$
- ▶ $b_j^{(l)}$ = bias for unit i in layer $l + 1$
- ▶ $a^{(l)}$ = activation (output value) of unit i in layer l .

Thus, $h_{\vec{w}, b}(x) = a_1^{(3)}$ which is:

$$a_1^{(3)} = f(w_{11}^{(2)} a_1^{(2)} + w_{12}^{(2)} a_2^{(2)} + w_{13}^{(2)} a_3^{(2)} + b_1^{(2)}) \text{ and}$$

$$a_1^{(2)} = f(w_{11}^{(1)} x_1 + w_{12}^{(1)} x_2 + w_{13}^{(1)} x_3 + b_1^{(1)})$$

Learning the net parameters: Stochastic Gradient Descent

Let's assume that the inputs are independent and identically distributed (I.I.D) in a distribution \mathcal{D} .

Define an error function $J(w, b; x, y)$ to determine how good is the approximation as

$$1/2 \|h_{w,b}(x) - y\|^2 + \lambda/2 \sum_l \sum_i \sum_j (w_{ij}^{(l)})^2 \quad (4)$$

First term controls the squared error, the second controls the weight decay to avoid overfitting.

Thus, we have to find w, b that minimise $E_{(x,y) \sim \mathcal{D}}[J(w, b; x, y)]$

Learning the net parameters: Stochastic Gradient Descent

while(stopcondition)

1. get next sample $(\vec{x}^{(i)}, y^{(i)})$
2. update $\vec{w}_j^l k = \vec{w}_j^l k - \alpha \frac{\partial J(w, b, \vec{x}^{(i)}, y^{(i)})}{\partial \vec{w}_j^l k}$
3. update $b_j^l = b_j^l - \alpha \frac{\partial J(w, b, \vec{x}^{(i)}, y^{(i)})}{\partial b_j^l}$

Improvements

- 1) The learning rate (α) can be dynamically adapted computing a momentum, see Adam algorithm.
- 2) We can determine the batch size to find a compromise between computing speed and data quality

Learning the net parameters: Backpropagation

How we can compute $\partial J(w, b, \vec{x}^{(i)}, y^{(i)})$ respect w and b ? \rightarrow using backpropagation.

Let δ_i^l be the parameter that determines how responsible is unit i in layer l of the error L . In the last layer $\delta_i^{(nl)}$ depends on $h_{w,b}$ and Y , this is $\delta_i^{(nl)} = -(y - a_i^{nl})f'(z_i^{(nl)})$, in the previous layers $l = nl - 1, \dots, 2$:

$$\delta_i^{(l)} = \left(\sum_j w_{ji}^{(l)} \delta_j^{(l+1)} \right) f'(z_i^{(l)}) \quad (5)$$

With δ known, we can update parameters with:

$$\begin{aligned} w_{ij}^{(l)} &= w_{ij}^{(l)} - \alpha (a_j^{(l)} \delta_i^{(l+1)} + \lambda w_{ij}^{(l)}) \\ b_i^{(l)} &= b_i^{(l)} - \alpha a_j^{(l)} \delta_i^{(l+1)} \end{aligned}$$

Thus, before each update step, we have to make a forward propagation to compute the loss and the activation functions a .

How to design an ANN?

We have the way to set the weights and bias but...

- ▶ Activation function (and its parameters)
- ▶ Number of units (width)
- ▶ Number of layers (depth)
- ▶ Regularisation (i.ex. dropout)
- ▶ hyperparameters initialization

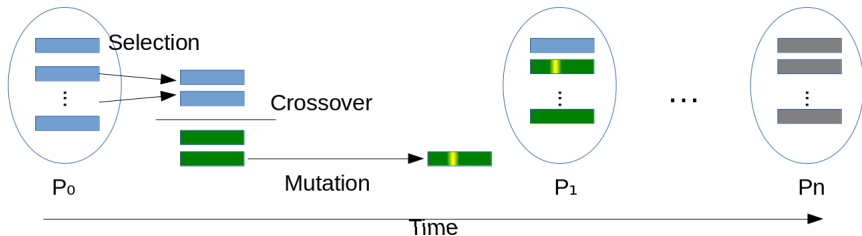
There is no closed form or optimal solution up to day, remains an "art"

Using a metaheuristic: Genetic algorithms

The number of layers and units are quite important: provide accuracy and generalisation

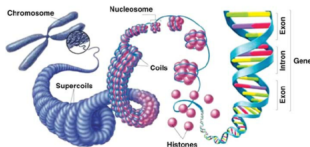
How to select them? Random? No, better to use evolution

Idea: the combination (crossover) of two good networks might improve the results



Using a metaheuristic: Genetic algorithms (II)

Nature

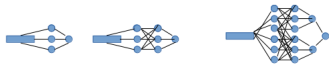


Artificial

Chromosome = [3,0,5,2,1]
Phenotype = Mean Squared Error
Genotype



Evolution through generations and time



Evolution through generations and time

Parameters that could be evolved

Number of layers, Number of units, Activation function, epochs, etc. → the GA needs its own parameters: operators, selection, individuals,...

2

²Images sources:

<https://www.researchgate.net/publication/292262701/figure/fig2/AS:323489350340609@1454137279164/Chromosomal-organization-of-a-eukaryotic-gene-in-which-exons-coding-regions-are.png>

<http://lindeyml.com/wp-content/uploads/2015/03/01-evolution-chain.jpg>

Using a metaheuristic: Genetic algorithms (III)

Other uses for these metaheuristics: dimensionality reduction

Consider the use of multiobjective optimisation (i.ex. small but significant number of variables)

Efficient implementation possibilities: intrinsically parallel Island model (with specialization) → Migration operator

Defining the problem

Given some simulations (CORSIKA), Can we build a model that identifies the type of particle?

Type of problem: what is identification? Regression multi-label classification multi-probability ensemble of binary classification

Other applications

Monitoring LATTES input!

Conclusions and Future Work

Conclusion

It is possible to apply ML successfully to some of the problems in astrophysics. However, a big advance could come dealing with unsupervised learning.

Future work : Autoencoders

Unsupervised learning (“blended” . . .) There are no y_i so we create them by $y^{(i)} = \vec{x}^{(i)}$, thus $h_{\vec{x},b}(x) = x$ Impose constrain that units should not activate, this is: $E_{\vec{x} \sim \mathcal{D}}[a_i^{(l)}] = \rho$ with $\rho \approx -1$ (inactive)
Therefore, the net will represent the distribution