

ALMA MATER STUDIORUM - UNIVERSITA' DI BOLOGNA  
CAMPUS DI CESENA  
SCUOLA DI SCIENZE

CORSO DI LAUREA IN INGEGNERIA E SCIENZE INFORMATICHE

CROSS-ORGANISM ANNOTATION PREDICTION THROUGH DEEP LEARNING  
ALGORITHMS

Relazione finale in

Programmazione di applicazioni Data Intensive

Relatore

Prof. Gianluca Moro

Co-Relatore

Dott. Ing. Roberto Pasolini

Presentata da

Marcello Feroce

Sessione 5 Ottobre 2017  
Anno Accademico 2016/2017

# Cross Organism annotation prediction through deep learning algorithms

Marcello Feroce

October 5, 2017

## Abstract

Studying how genes or proteins influence humans and other species' lives is paramount. To study that, it's necessary to know which functional properties are specific for each gene or protein. The association between one gene or protein and a functional property is called annotation. An annotation can be 0 or 1. 1 means that gene or protein contributes to the activation of a certain functional property. Functional properties are referred by terms, which are strings that belong to ontologies. This work aim is to predict novel gene annotations for little know species such as *Bos Taurus*. To predict such annotations, a model, built using deep learning, is used. This model is trained using well know species as *Mus Musculus* or *Homo Sapiens*. Every predicted annotation has its own likelihood, that tells about how much the prediction is close to a 0 or a 1. Final accuracy can be evaluated fixing a certain value of likelihood, so that all the considered annotations have a likelihood greater or equal than the fixed one. The obtained accuracy is quite high but not enough to be used in a professional way, although it offers a nice cue for future research.

# Contents

|   |           |
|---|-----------|
| <b>1 Introduction</b>                                 | <b>2</b>  |
| 1.1 Domain of the work . . . . .                      | 2         |
| 1.2 Aim of the work . . . . .                         | 6         |
| <b>2 Annotation prediction in literature</b>          | <b>7</b>  |
| <b>3 Methods</b>                                      | <b>11</b> |
| 3.1 Machine learning . . . . .                        | 12        |
| 3.2 Deep learning . . . . .                           | 13        |
| 3.2.1 Neural Networks . . . . .                       | 13        |
| 3.2.2 Regression and deep learning . . . . .          | 16        |
| 3.2.3 Supervised learning and deep learning . . . . . | 16        |
| 3.2.4 Training a neural network . . . . .             | 16        |
| <b>4 Formalisation of the problem</b>                 | <b>20</b> |
| 4.1 Costs . . . . .                                   | 22        |
| <b>5 Solutions and choices</b>                        | <b>24</b> |
| 5.1 Choice of neural network type . . . . .           | 24        |
| <b>6 Technologies</b>                                 | <b>27</b> |
| <b>7 Experimental results</b>                         | <b>28</b> |
| <b>8 Conclusions</b>                                  | <b>34</b> |
| <b>9 Future developments</b>                          | <b>36</b> |
| <b>10 Appendix</b>                                    | <b>37</b> |
| 10.1 Implementation of the software . . . . .         | 37        |
| 10.2 Usage of the software . . . . .                  | 40        |

# 1 Introduction

Knowledge of how structural or functional biological property of a living being can differ from being to being has a primary importance. It helps understanding physiological and pathological biological processes, as well as developing new drugs and therapies.

Aim of this work is to predict novel structural or functional biological properties of little known organisms, which are little studied by biologists, for several reasons. Functional biological properties, which can be referred by terms, short unique strings basically, tell us a lot about how a living being can differ from another. One example can be the ability of a cell to reproduce itself. These functional biological properties are depending on genes and proteins(which depend on genes). So, being able to predict functional biological properties, could simplify a lot the work of biologists, and understanding how a living being behaves, depending on its genes would be much faster than it is nowadays. Of course a very high accuracy is necessary, and it has to be obtained in a relatively short time.

Next sections will show in a more accurate way all the steps taken to get the results. Starting with an introduction to the domain, annotation prediction in literature will follow. An explanation of Machine Learning and Deep Learning is necessary then of course before starting explaining in a more specific way how the prediction works. Indeed the algorithm used and the choices are explained, followed by the technologies used. Finally Experiments and results are exposed.

## 1.1 Domain of the work

To have a clearer idea on how the project works it's necessary to have clarify some basic concepts.

An organism has got many biomolecular entities, which are

included in all the possible biomolecular entities of its species. Biomolecular entities are mainly genes and protein products. Species have also a set of possible structural or functional biological properties which can be working or not. A property can be part of a subset of properties, or be itself a subset of properties, thus containing different properties which all respect its function. Making an instance to understand better, the property "having a volume enclosed by the nuclear inner membrane" is part of the property "having any constituent part of the nucleus, a membrane-bounded organelle of eukaryotic cells in which chromosomes are housed and replicated" but also of the property "having an organelle lumen that is part of an intracellular organelle".

Biomolecular entities are identified by an abbreviation. Structural or functional biological properties refer to terms, which are part of ontologies. Ontologies contain terms and define their structure through a Directed Acyclic Graph (DAG). The Gene Ontology (GO) is the most considerable. The DAG shows the hierarchies of the terms with IS\_A and PART\_OF relationships. The schema of the DAG would look as a tree. A controlled biomolecular annotations (often referred just as annotations) is an association between a biomolecular entity and a term in an ontology.

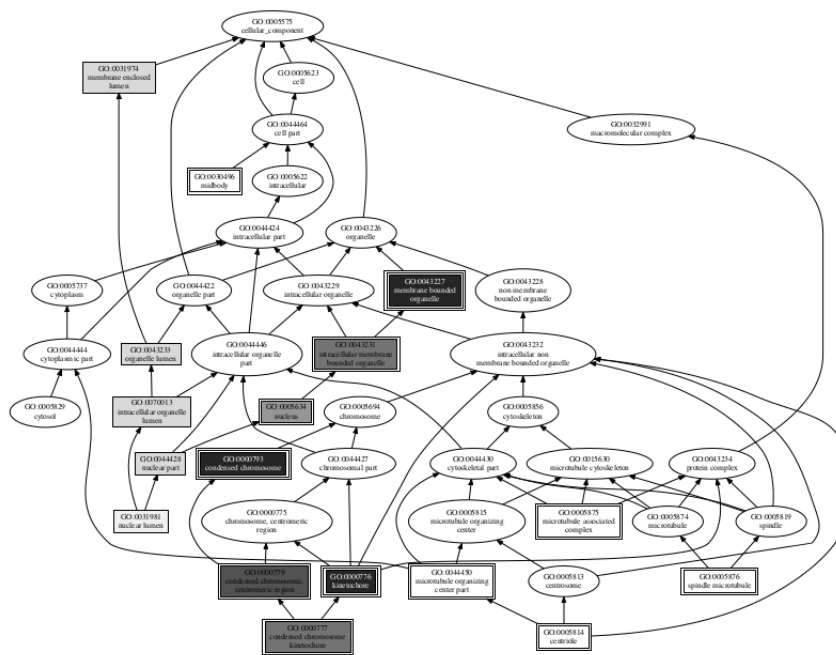


Figure 1: Representation of a DAG. Got from this work[1]

An annotation matrix  $A(g, t)$  is a binary matrix whose  $i$ -th row  $A(g_i, t)$  is the annotation profile of the gene  $g_i$  to the terms  $t \in T$ ; thus, a binary gene-term annotation matrix  $A(g, t)$  represents a set of known annotations of some genes  $g$  to some controlled terms  $t \in T$ . This means that there could be several annotation matrix for each species, having different subsets of biomolecular entities and terms, and different values of the annotations, depending on the year and on the research.

Table 1: example of GO term

|            |   |
|------------|---|
| id:        | GO:0000016  |
| name:      | lactase activity  |
| namespace: | molecular_function  |
| def:       | "Catalysis of the reaction: lactose + H2O = D-glucose + D-galactose." |
| synonym:   | "lactase-phlorizin hydrolase activity" BROAD [EC:3.2.1.108]           |
| synonym:   | "lactose galactohydrolase activity" EXACT [EC:3.2.1.108]              |
| xref:      | EC:3.2.1.108  |
| xref:      | MetaCyc:LACTASE-RXN   |
| xref:      | Reactome:20536  |
| is_a:      | GO:0004553 ! hydrolase activity, hydrolyzing O-glycosyl compounds     |

Each annotation in the annotation matrix is represented by 1 either a 0. If it's 1, it means that the biomolecular entity in that row activates (by itself or with other biomolecular entities) that term in that column.

There are some annotation matrices, called Inferred from Electronic Annotation (IEA), whose annotations are computed. These ones are as not reliable as the normal ones, but they can be useful anyway.

Some biomolecular entities can be included in more annotation matrices of different species, since more species can have part of biomolecular entities which are in common. The same speech could be done for terms: Different species can have same properties.

| No. | gene<br>string | GO:0005737<br>Normal | GO:0044255<br>Normal | GO:0006807<br>Normal | GO:0080090<br>Normal | GO:0010628<br>Normal | GO:0019899<br>Normal | GO:0044424<br>Normal | GO:0006575<br>Normal | GO:0010504<br>Normal | GO:0044262<br>Normal | GO:0003674<br>Normal | GO:0032386<br>Normal | GO:0019219<br>Normal | GO:0005488<br>Normal |
|-----|----------------|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|
| 1   | ACADM          | 0                    | 1                    | 0                    | 0                    | 0                    | 0                    | 0                    | 0                    | 0                    | 0                    | 1                    | 0                    | 0                    | 0                    |
| 2   | MAP1LC3A       | 1                    | 0                    | 0                    | 0                    | 0                    | 0                    | 1                    | 0                    | 0                    | 0                    | 1                    | 0                    | 0                    | 1                    |
| 3   | SMUG1          | 0                    | 0                    | 1                    | 0                    | 0                    | 0                    | 1                    | 0                    | 0                    | 0                    | 1                    | 0                    | 0                    | 0                    |
| 4   | MTIF2          | 1                    | 0                    | 0                    | 1                    | 0                    | 0                    | 1                    | 0                    | 0                    | 0                    | 1                    | 0                    | 0                    | 1                    |
| 5   | ERCC2          | 0                    | 0                    | 1                    | 1                    | 1                    | 0                    | 1                    | 0                    | 1                    | 0                    | 1                    | 0                    | 1                    | 1                    |
| 6   | GNAT1          | 0                    | 0                    | 0                    | 0                    | 0                    | 1                    | 1                    | 0                    | 0                    | 0                    | 1                    | 0                    | 0                    | 1                    |
| 7   | BGN            | 1                    | 0                    | 0                    | 0                    | 0                    | 0                    | 1                    | 0                    | 0                    | 0                    | 0                    | 0                    | 0                    | 0                    |
| 8   | ABAT           | 1                    | 1                    | 1                    | 0                    | 0                    | 0                    | 1                    | 1                    | 0                    | 0                    | 1                    | 0                    | 0                    | 1                    |
| 9   | GJA1           | 1                    | 0                    | 0                    | 1                    | 0                    | 0                    | 1                    | 0                    | 1                    | 0                    | 1                    | 0                    | 0                    | 1                    |
| 10  | BCKDK          | 1                    | 0                    | 1                    | 0                    | 0                    | 0                    | 1                    | 0                    | 0                    | 1                    | 1                    | 0                    | 0                    | 1                    |
| 11  | MOC53          | 1                    | 0                    | 1                    | 0                    | 0                    | 0                    | 1                    | 0                    | 0                    | 0                    | 1                    | 0                    | 0                    | 0                    |
| 12  | CRYAB          | 1                    | 0                    | 0                    | 0                    | 0                    | 0                    | 1                    | 0                    | 0                    | 0                    | 1                    | 1                    | 0                    | 1                    |
| 13  | ERCC8          | 0                    | 0                    | 1                    | 1                    | 0                    | 0                    | 0                    | 0                    | 1                    | 0                    | 1                    | 0                    | 1                    | 1                    |
| 14  | FUS            | 1                    | 0                    | 0                    | 1                    | 1                    | 0                    | 1                    | 0                    | 1                    | 0                    | 1                    | 0                    | 1                    | 1                    |
| 15  | EIF3K          | 1                    | 0                    | 0                    | 0                    | 0                    | 0                    | 1                    | 0                    | 0                    | 0                    | 1                    | 0                    | 0                    | 1                    |
| 16  | BCL10          | 1                    | 0                    | 0                    | 1                    | 1                    | 1                    | 0                    | 1                    | 0                    | 1                    | 0                    | 1                    | 1                    | 1                    |
| 17  | ERCC3          | 0                    | 0                    | 1                    | 1                    | 1                    | 0                    | 1                    | 0                    | 1                    | 0                    | 1                    | 0                    | 1                    | 1                    |
| 18  | HIF1A          | 0                    | 0                    | 0                    | 1                    | 1                    | 1                    | 1                    | 0                    | 1                    | 0                    | 1                    | 0                    | 1                    | 1                    |
| 19  | SCARAS         | 0                    | 0                    | 0                    | 0                    | 0                    | 0                    | 0                    | 0                    | 0                    | 0                    | 1                    | 0                    | 0                    | 0                    |
| 20  | FRA35          | 1                    | 1                    | 0                    | 0                    | 0                    | 0                    | 1                    | 0                    | 0                    | 0                    | 1                    | 0                    | 0                    | 0                    |
| 21  | SDD3           | 1                    | 0                    | 1                    | 1                    | 0                    | 1                    | 1                    | 1                    | 0                    | 1                    | 0                    | 0                    | 0                    | 1                    |
| 22  | MOCOS          | 0                    | 0                    | 0                    | 0                    | 0                    | 0                    | 0                    | 0                    | 0                    | 0                    | 1                    | 0                    | 0                    | 0                    |
| 23  | IDH1           | 1                    | 0                    | 0                    | 0                    | 0                    | 0                    | 1                    | 1                    | 0                    | 0                    | 1                    | 0                    | 0                    | 0                    |
| 24  | MVOD1          | 0                    | 0                    | 0                    | 1                    | 0                    | 0                    | 1                    | 0                    | 0                    | 0                    | 1                    | 0                    | 1                    | 1                    |
| 25  | MAEL           | 1                    | 0                    | 1                    | 0                    | 0                    | 0                    | 1                    | 0                    | 0                    | 0                    | 0                    | 0                    | 0                    | 0                    |
| 26  | RCP9           | 0                    | 0                    | 1                    | 0                    | 0                    | 0                    | 1                    | 0                    | 0                    | 0                    | 1                    | 0                    | 0                    | 0                    |
| 27  | ILL2A          | 0                    | 0                    | 0                    | 0                    | 0                    | 0                    | 0                    | 0                    | 0                    | 0                    | 1                    | 0                    | 0                    | 1                    |
| 28  | PSEN1          | 1                    | 0                    | 0                    | 0                    | 0                    | 0                    | 1                    | 0                    | 0                    | 0                    | 0                    | 0                    | 0                    | 0                    |
| 29  | GAPVD1         | 1                    | 0                    | 0                    | 0                    | 0                    | 0                    | 1                    | 0                    | 0                    | 0                    | 1                    | 0                    | 0                    | 1                    |
| 30  | NFYA           | 0                    | 0                    | 0                    | 1                    | 0                    | 0                    | 1                    | 0                    | 0                    | 0                    | 1                    | 0                    | 1                    | 1                    |
| 31  | TLR9           | 0                    | 0                    | 0                    | 0                    | 0                    | 0                    | 0                    | 0                    | 1                    | 0                    | 1                    | 0                    | 0                    | 1                    |
| 32  | JUP            | 1                    | 0                    | 0                    | 0                    | 0                    | 0                    | 1                    | 0                    | 0                    | 0                    | 0                    | 0                    | 0                    | 0                    |
| 33  | ROMO1          | 1                    | 0                    | 0                    | 0                    | 0                    | 0                    | 1                    | 0                    | 0                    | 0                    | 0                    | 0                    | 0                    | 0                    |
| 34  | SLC7A5         | 0                    | 0                    | 0                    | 0                    | 0                    | 0                    | 0                    | 0                    | 0                    | 0                    | 1                    | 0                    | 0                    | 0                    |
| 35  | GHR            | 0                    | 0                    | 0                    | 0                    | 0                    | 0                    | 0                    | 0                    | 0                    | 0                    | 1                    | 0                    | 0                    | 1                    |
| 36  | CI GALT1       | 0                    | 0                    | 0                    | 0                    | 0                    | 0                    | 0                    | 0                    | 0                    | 0                    | 1                    | 0                    | 0                    | 0                    |
| 37  | AHCY           | 1                    | 0                    | 1                    | 0                    | 0                    | 0                    | 1                    | 1                    | 0                    | 1                    | 0                    | 0                    | 0                    | 1                    |

Figure 2: Example of annotation matrix

In general, annotation matrices have thousands of genes and terms, and are sparse matrices, i.e. matrices in which most of the elements are zero.

## 1.2 Aim of the work

As introduced at the beginning of the chapter, prediction is the mainly thing this project has to do. Having clarified some concepts in the previous section allows to explain better the aim of the work.

Having some genes of a little know species, thus with a very few known annotations, there should be a program which returns the annotations associated with those genes for a certain set of terms. Trying to predict novel gene annotations, there are several methods that can do that or at least give a contribute. In this work a neural network, which is the model used in deep learning, a sub-type of machine learning, has been used. Several other methods can be used, but working with neural network could be a good idea, after the good results obtained in this work [1] with machine learning. In this method there is a preliminary work that has to be



done previously of building the model. This preliminary work is in a certain way similar to the one exposed in the method used in this work [1], that is the one this work is inspired by. This preliminary work's aim is fundamentally to get the right and best annotations that will be used to train and use the model. In the field of bioinformatics, annotation prediction is very useful, but it's only one of the possible aims that bioinformatics has. Some of them are completely different, but many others are very related to annotation prediction, and can be useful in annotation prediction. Other times instead, annotation prediction is attempted, but with completely different approaches.

## 2 Annotation prediction in literature

Predicting novel gene annotations has a primary importance, as told before. It comes hard in general to predict them in a short way with even a great accuracy. As we will see flexibility and speed go to the opposite way than valuable prediction.

Many computational approaches have been proposed to find out novel annotations of biomolecular entities through the processing of different kinds of experimental data as biomolecular sequences, gene expressions, protein interactions and phylogenetic profiles, worked out using different techniques such as sequence homology, network based or data and text mining based methods [38][39]. Multiple types of data have been approached in several tries, even from different species through comparative genomic approaches. These attempts pursued to improve results(e.g., [41][40]). These approaches and methods are usually pretty complex and very costly, and didn't show great results.

Many different methods or approaches have been tried in order to predict new biomolecular annotations through compu-

tational techniques:

A latent semantic approach to build a prediction algorithm based on the Singular Value Decomposition (SVD) method with gene-to-term annotation matrices [6] [7]. The algorithm basically counts co-occurrences between pairs of annotation terms. As in the work in [8] this algorithm was improved. They computed gene functional similarity on the GO annotations of the genes and used it to include gene clustering. Then, they extended it, being able to automatically choose the best SVD truncation level. The SVD has also been used with annotation weighting schemes [9] [10] [11], built on term and gene frequencies. Although very adaptable on the organism and the term vocabulary, these methods brought limited accuracy.

As in this work [42], a prototypal multi-organism Genomic and Proteomic Data Warehouse, called GPDW, was built, to predict gene annotations basing on on information and data integration, using SVD, and SIM(Semantic improvement), but they didn't get excellent results.

In this work [43] researchers worked on Protein Annotation Prediction using Metric Labeling and Semi-metric Embedding. They got quite good results, although their method didn't show robustness.

Sophisticated latent semantic analysis techniques, mainly related to Latent Semantic Indexing (LSI) [12] have been proposed to predict biomolecular annotations on the basis of available annotations leveraging on available annotations. These were formerly used for Natural Language Processing. These techniques include the probabilistic Latent Semantic Analysis(pLSA) [13], which uses the latent model of a set of annotation terms to improve reliability of annotation prediction

results. This technique was used in [14] and was improved with weighting schemes[15]. This brought an improvement comparing to the SVD method in this work [16].

Latent Dirichlet Allocation (LDA) algorithm has been at the basis of topic modeling in this work[17]. LDA was also used in two works[18][19] to separate gene expression microarray data in clusters. LDA algorithm associated with the Gibbs sampling[21][22][23] was improved to predict gene annotations in this work[20]. Despite these complex techniques are far better than the ones based on linear algebra, these are not good when the data size increase due to slowness.

Also biological network analysis is frequently used to predict gene functions:

As in the work in[24], GeneMANIA was built. GeneMania is a server whose purpose is gene function prediction. GeneMania takes the inputs(query gene sets) as networks whose associated weight is based on the strength of the connectivity between the genes in the query set, compared with their connectivity to non-query genes.

As in the work in [25] reserachers instead leveraged on a kernel-based learning method whose approach is based on a labeled graph kernel which can predict functions of single genes basing on the function distributions which can be evaluated from their associated gene interaction networks.

Tecniques from Hidden Markov Model(HMM) were also used to get a model of the evolution of genes[26], or to predict gene function basing on sequential gene expression data[27]

In this work[28] there was a proposal to use decision trees

and Bayesian networks to predict annotations by learning patterns from available annotations.

In the work in [29], researchers suggested a k-nearest neighbour (k-NN) classifier to associate a gene with new annotations common among its functionally nearest neighbour genes. In this method gene functional distance is computed according to the semantic similarity of the GO terms that annotate the genes.

Gene function prediction techniques use also a lot of Support Vector Machine (SVM) classifiers:

SVM classifiers were used in this work [30] to predict annotations for many eukaryotic protein sequences. Another work [31] instead leveraged them to predict potential functions for previously unannotated *Drosophila melanogaster* genes, analysing a huge dataset from gene expression microarray experiments.

New supervised methods have come recently, even for annotation prediction. Two examples are the works [32]. and [33].. The first one considered the prediction of gene functions as a multi-label top-down classification problem. This method is based on hierarchical relationships in the GO structure that soften the quantitative difference in quantity, between negative and positive samples used for training. Cheng et al.'s work showed low accuracy.

The second one suggested that relations between genes generate autocorrelation in gene annotations and breach the assumption that annotations are independently and identically distributed. Stojanova et al. got better a slightly better accuracy.

Another used solution to predict novel gene annotations is to use multiple data types or sources, even from different

species. Several experiments have been done. An example is this work [34], where researchers built a general Bayesian framework to integrate heterogeneous types of high-throughput biological data. The Bayesian framework was applied for the prediction of *Saccharomyces cerevisiae* gene functions.

a SVM classifier for each gene annotation to a GO term was trained in this work[35]. using gene expression levels from microarray experiments. Further consistency among predicted annotation term was enforced by means of a Bayesian network mapped on the GO structure.

Differently, text mining techniques were used in this work[36] and in this other work[37] to extract from the literature gene associated keywords which are then mapped to GO concepts.

These approaches got better results with compared to similar methods applied on a single data type but, a preparatory data integration step is needed, and this adds complexity, decrease flexibility and slows the prediction process. Thus, formerly suggested methods for biomolecular(gene or protein) annotation prediction either are general and flexible, but show low accuracy, or get better results by either working with a complex integrative analytical framework or adopting a more complex model. The reason for the low accuracy is mainly given from the simple model used. On the other hand, leveraging a complex integrative analytical framework is often hard and takes a long time to be set up. Using a more complex model instead heavily slows the prediction process, in particular when the testing data size increases a lot.

### **3 Methods**

A model is needed to predict novel gene annotations. This model has to be built with annotation matrices of very well known(and so studied) species. The model would show all its power working with annotation matrices of little know species

as input.

### **3.1 Machine learning**

Not always a program runs a clear and procedural algorithm written by someone. In the cases it doesn't, sometimes it's possible to say that the programs uses machine learning algorithms. Machine learning consists in a self-made learning of a knowledge model from a huge amount of example data, called training set. Machine learning build the aforesaid model generalising patterns observed in the training set.

Unlike the usual way to program, in machine learning, programs are taught how to process data, from examples.

Lots of approaches can be used with machine learning. Some of them are Decision tree learning, Association rule learning, Deep learning.

Each of these approaches can address a wide variety of tasks as:

- Classification
- Regression: Find a pattern between variables
- Clustering: Unlike classification, the groups are not known beforehand
- Density estimation: Finds the distribution of inputs in some space
- Dimensionality reduction: Maps the input into a lower-dimensional space

Tasks can also be classified in another way. Usually they are split in three large categories, depending on the "feedback" received during the learning:

- Supervised learning
- Unsupervised learning

- Reinforcement learning

## 3.2 Deep learning

Deep learning is a subtype of machine learning whose model is a neural network

### 3.2.1 Neural Networks

Neural networks are the models used by deep learning. Neural networks, whose name actually should be "Artificial neural networks", have this particular name because they are inspired by the biological neural networks that constitute animal brains. Neural networks can implement many tasks among the ones of machine learning. Neural networks are very used to implement regression and Classification.

Each neural network consists in one input layer, one output layer, and an indefinite number of hidden layers, which could be even 0. However usually the number of layers goes from two to four. Each layer includes nodes. In the input layer nodes consist in the inputs given to the model.

Each layer can have a different number of nodes.

Between two layers there are other two components:

- Weights
- Biases

The number of weights between two layers corresponds to the number of nodes in the first layer(the one closer to the input layer) multiplied by the number of the nodes in the second layer. Indeed, for each couple of nodes there is a weight.

The number of biases between two layers corresponds to the number of nodes in the second layer(the one closer to the output layer).

The value of each node of the next layer is given by this easy procedure:

- Multiply each node in input with its weight that refers to that node in output
- Sum the previously obtained values
- Sum an offset value, called bias, to the value previously obtained
- Apply to the obtained value, a function, called activation function

This results that the vector of outputs  $y$  is given by:

$$y = \sigma\left(\sum_j W_{i,j}x_j + b_i\right) \quad (1)$$

Where  $W$  is the matrix of weights;  $x$  is the vector of inputs, and  $\sigma$  is the activation function. Sometimes the activation function takes an array as input. In this case of course the function is not applied value by value, but it's applied to the whole vector obtained after applying weights and biases.

The softmax function is often used as activation function, mostly for classification task. The softmax function takes as input a vector and returns a vector of the same shape with the features that each value is between 0 and 1; the sum of all the values in the vector is 1.

Softmax function:

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^K e^{z_k}} \quad \forall j = 1, \dots, K \quad (2)$$

Other functions are for instances:

- Rectifier
- Sigmoid

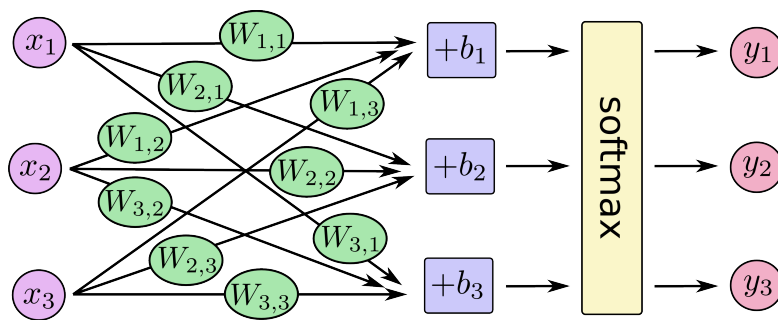


- Sin

The purpose of training the neural network is to get the best weights and biases in order to have the output of the neural network and the desired output as close as possible.

Here down there is an example of how every pair of layers works. In this example the activation function used is the softmax function, there are three nodes in input, and three nodes in output.

It's possible to see how the vector of outputs  $y$  can be calculated just multiplying the matrix of weights by the vector of inputs, then summing it to the vector of biases, and finally apply the activation function.



$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \text{softmax} \left( \begin{bmatrix} W_{1,1}x_1 + W_{1,2}x_2 + W_{1,3}x_3 + b_1 \\ W_{2,1}x_1 + W_{2,2}x_2 + W_{2,3}x_3 + b_2 \\ W_{3,1}x_1 + W_{3,2}x_2 + W_{3,3}x_3 + b_3 \end{bmatrix} \right)$$

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \text{softmax} \left( \begin{bmatrix} W_{1,1} & W_{1,2} & W_{1,3} \\ W_{2,1} & W_{2,2} & W_{2,3} \\ W_{3,1} & W_{3,2} & W_{3,3} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \right)$$

$$y = \sigma \left( \begin{bmatrix} W_{1,1} & W_{1,2} & \cdots & W_{1,n} \\ W_{2,1} & W_{2,2} & \cdots & W_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ W_{m,1} & W_{m,2} & \cdots & W_{m,n} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix} \right) \quad (3)$$

### 3.2.2 Regression and deep learning

Regression analysis is a process that estimates the relationships among variables. Regression studies the relations between a dependent variable and a set of independent variable. Regression aim is so to build a model which allows to do predictions. Of course the model will change during this process, changing its parameters. The easiest regression is the linear regression, which is the one used in deep learning. In linear regression the function that is used by the model is linear.

Regression is used a lot for predictions. In fact in deep learning, neural networks through their layers, and through the formulas previously explained, which are linear, can predict one value, giving them an input.

### 3.2.3 Supervised learning and deep learning

Supervised learning is a particular task of machine learning. In Supervised learning the model is given a set of desirable outputs, called often labels, for each iteration. The model trains, trying to get as close as possible to the labels. During the training a set of inputs with the corresponding labels is given to the neural network so that the network can find a pattern and obtain the best weights and biases.

### 3.2.4 Training a neural network

A neural network has to be built in order to get the right weights and biases.

To reach this target the backpropagation method and gradient descent method is used.

Actually gradient descent method is used by backpropagation method[5], in order to minimise the error, between the desired output and the computed one. Indeed, the gradient descent method aim is to find the minimum of a function.

In the gradient descent, if the function is differentiable nearby a certain point  $a$ , then the function decreases fastest if one goes from  $a$  in the direction of the negative gradient of the function at  $a$ ,  $-\nabla F(a)$ .

It's then possible to go on with this iterative algorithm, finding another point where to go on with the method, until the minimum is found.

The consequences of these statements are are that

$$\mathbf{a}_{n+1} = \mathbf{a}_n - \gamma \nabla F(\mathbf{a}_n)$$

$\gamma$  called step size, is a scalar value that represents the "length" of the step. The step size can change at every iteration.

The backpropagation method uses the gradient descent to minimise the error function. The error function can be represented for instance by the mean squared error function:

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^n (\hat{Y}_i - Y_i)^2$$

Starting from a generic point, the algorithm starts computing the gradient. Then, using the gradient descent, finds the minimum of the function of the error.

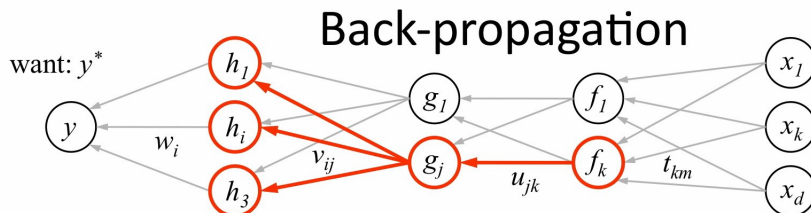
The backpropagation algorithm can be divided into two steps:

- Forward pass: the input given to the network is propagated to the next level and so on to the next levels(the information flow moves forward). the error is so computed.

- Backward pass: the error done by the network is propagated backward and the weights and the biases are updated in the right way.

The logic steps to train a neural network with supervised learning are the following ones:

- Feed the network with inputs and labels
- Initialising the weights and the biases
- Begin the cycle composed of forward and backward passes
  - Applying the network to the given inputs, computing the outputs(forward pass, from input layer to output layer)
  - Compute the error through the error function, the outputs and the labels
  - Through the gradient descent, correct the weights and the biases(backward pass, from output layer to input layer)



1. receive new observation  $\mathbf{x} = [x_1 \dots x_d]$  and target  $y^*$
2. **feed forward:** for each unit  $g_j$  in each layer  $1 \dots L$  compute  $g_j$  based on units  $f_k$  from previous layer:  $g_j = \sigma \left( u_{j0} + \sum_k u_{jk} f_k \right)$
3. get prediction  $y$  and error  $(y - y^*)$
4. **back-propagate error:** for each unit  $g_j$  in each layer  $L \dots 1$

|   |  |
|---|--|
| <p>(a) compute error on <math>g_j</math></p> $\frac{\partial E}{\partial g_j} = \sum_i \underbrace{\sigma'(h_i)}_{\text{how } h_i \text{ will change as } g_j \text{ changes}} v_{ij} \underbrace{\frac{\partial E}{\partial h_i}}_{\text{was } h_i \text{ too high or too low?}}$ <p>should <math>g_j</math> be higher or lower?</p> | <p>(b) for each <math>u_{jk}</math> that affects <math>g_j</math></p> <p>(i) compute error on <math>u_{jk}</math>      (ii) update the weight</p> $\frac{\partial E}{\partial u_{jk}} = \frac{\partial E}{\partial g_j} \underbrace{\sigma'(g_j)}_{\text{do we want } g_j \text{ to be higher/lower}} f_k$ <p>how <math>g_j</math> will change if <math>u_{jk}</math> is higher/lower</p> $u_{jk} \leftarrow u_{jk} - \eta \frac{\partial E}{\partial u_{jk}}$ |
|---|--|

Copyright © 2014 Victor Lavrenko

There are some problems that can occur while training a neural network.

Overfitting:

Overfitting is a common problem that affects machine learning. It happens when the used model fits too well with the training data, thus missing the aimed and expected generalization. It causes a worse accuracy with the test data. Overfitting can be caused by a too small or too big training

Vanishing gradient:

The vanishing gradient problem is particular of gradient based methods. One of this gradient based methods is the backpropagation method.

In this problem the gradient tends to get smaller as we move backward through the hidden layers, and this phenomenon of course is stronger on earlier layers.

Give a function  $f(x)$ , if  $f'(x)$  is small, it means  $x$  is close to the minimum. In a similar way, minimising the error function in backpropagation, the small gradient in the early layers might mean not much adjustment of the weights and biases is needed.

This is not the case, otherwise it wouldn't be a problem.

Vanishing gradient happens because the cost is dependent on the derivatives of the result of the activation function for each layer.

The sigmoid function, which always gives an output between 0 and 1 is subject to this problem indeed. A product of lower than 1 values, gives a much lower value.

This problem is more likely to happen as more layers are added to the neural network.

## 4 Formalisation of the problem

The algorithm used to try predict novel gene annotations, is based on neural networks.

But first, before applying the annotations to the networks there are some things that have been done:

- To choose the annotation matrices to be used at least two annotation matrices of the same species from studies of different times are needed. Once chosen the ages to train and use the model for, every matrix used has to be aged as the same age of the studies.
- To take just a partial set of the chosen annotation matrices for the training, having chosen a integer positive value  $M$ 
  - Take all the annotations that refer to the common terms between the source old and the target old annotation matrices
  - From the previous annotations, to take just the annotations for those genes which have at least  $M$  true(1) annotations

Then train the neural network giving inputs with a shape of a list of annotations for a gene. The output has the same shape, and each predicted annotation has to be a decimal.

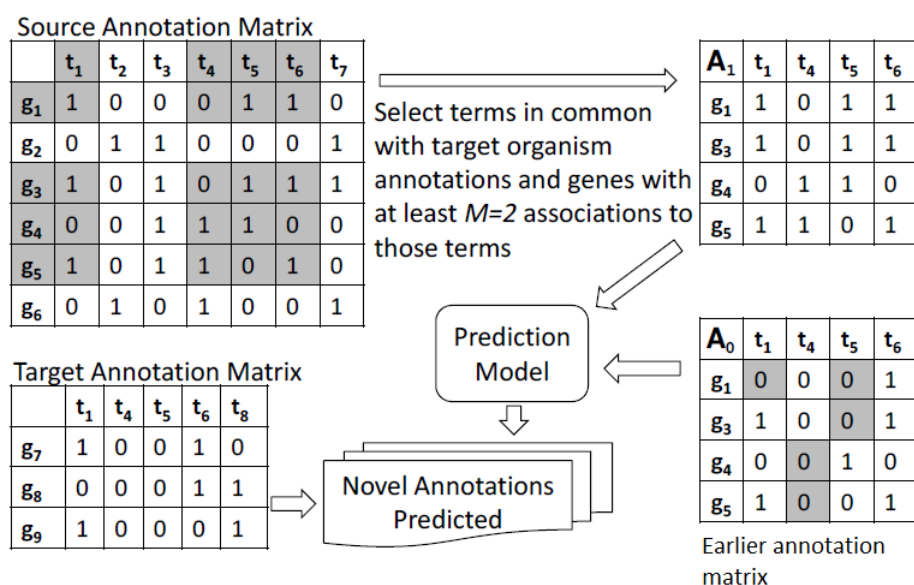


Figure 3: a schematic figure of the method. Got from this work [1]

The used annotation matrices have to be subjected to a process called unfolding. This process consists in:

- Taking a list of lists, from a file, called "ancestors file". For each first term of every list the other ones of their lists represent their ancestors in the DAG
- For each corresponding annotation of each first term in every list:  
If the annotation is true(1) for a certain gene, then all the annotations for that gene that correspond to the ancestors of the terms have to be set to 1

Then to measure the accuracy, having set a tolerance float value between 0 and 1:

- Correct each annotation likelihood following the hierarchy of the terms. If a predicted annotation is lower than 0.5, then its likelihood is given by 1 minus its value. Otherwise

its likelihood corresponds to its value. This correction process has to be done in this way:

For each novel gene-term annotation, its likelihood has to be increased with the average of the likelihoods of all the annotations of the gene to all the ancestors of the term and then normalising such hierarchical likelihood, as follows:

$$p^h(g,t) = \frac{\frac{\sum_{t_a \in \text{ancestors}(t)} p(g,t_a)}{|\text{ancestors}(t)|} + p(g,t)}{2} \quad (4)$$

- From all the predicted annotations, select only the ones that have likelihood greater than the tolerance.
- Let's say a predicted annotation is correct if it's lower than 0.5 and it's label is 0, or if it's greater or equal than 0.5 and its label is 1
- If a predicted annotation is not correct using the label annotation matrix, use if existing the IEA matrix of that species for that year to find if that annotation is correct
- The accuracy is given by the number of correct predicted annotations divided by the number of the ones with a likelihood greater or equal than the set tolerance value

## 4.1 Costs

All the preparatory processes needed to get the data to train the network, the training itself, and the evaluation of the accuracy have a pretty high cost that involves a lot of time and resources.

To get the common terms from several lists of terms:



First select the shortest list among the given ones.

Then scan each term of the shortest list, and verify if that term is included in all the other list. If a term is not included in at least one of the given lists, the scanning passes to the next term. Otherwise, the term is added to the list of common terms, that is returned at the end.

This process is  $O(n)$ , where  $n$  is the average number of terms of each list.

To get the list of genes that will train the neural network, given the  $M$  factor, the common terms, and the two annotation matrices of the same species of different ages:

Scan every gene of the older matrix. If that gene is included in the earlier annotation matrix go on. Then if the number of true annotations is greater or equal than  $M$ , counting only the annotation that refer to the common terms, add that gene to a list of gene that will be returned.

Getting all the gene of a matrix is  $O(m)$ , where  $m$  is the number of genes of a matrix.

Then counting the annotations in order to compare that value with  $M$  is  $O(n)$  where  $n$  is the number of terms of an annotation matrix. Thus the cost of this method is  $O(m*n)$ .

Actually to respect the algorithm just the older one could be used, but even the earlier one is needed because not all the genes in the older version are included in the earlier one.

Then to correct the predicted annotations using the hierarchy of the terms, referring to (4) formula to get the correct likelihood of a predicted annotation, it's necessary to:

First scan all the annotations of the predicted matrix. Then for each annotation, look for ancestor terms of the term that refers to that annotation. All this means a very heavy work. Indeed just scanning all the annotations is  $O(m*n)$ , with  $n$  number of terms, and  $m$  number of genes. Then, if a term that refers to an

annotation has ancestors, it's necessary to scan all the ancestors, which are in an average number "t".

In conclusion, this work is very computationally costly. It is  $O(m*n*t)$ .

Finally, to evaluate the accuracy of the predicted annotations, it's necessary to scan all the annotations of the predicted matrix. This means this work is  $O(m*n)$

## 5 Solutions and choices

To choose the annotation matrices to be used it's better to train the model with annotation matrices of the same well known species. Then it makes more sense to use the model with annotation matrices of a little known species.

The old annotation matrix of the well known species will work as input during the training; the more recent one will work as label. Then the old annotation matrix of the little known species will be used as input for the testing, and the more recent one(of the same species) will be used to test the accuracy In order to train the network in a good way, as in the work in[4] tells, it's better to choose a fixed batch list of items that will train the network, of a certain batch size. This list has to be shuffled at every iteration(epoch) of training.

For instance, on 500 genes that have been selected as good for training, just 10 or 20 can be selected to proper train the neural network.

### 5.1 Choice of neural network type

Neural networks can be divided in some types. The main ones are:

- Feed forward neural networks, which can be divided into:
  - Dense neural network: Where all its layers are dense
  - Convolutional neural network: Where at least one convolutional or pooling layer is included
- Recurrent neural networks, whose layers maintain an internal state
- Long Short Term Memory neural networks, whose layers maintain an internal state through cells and gates
- Neural Turing Machines, which combine neural networks with turing machines, which have an addressable memory

A layer is dense if it has as many nodes as the previous layer.

A layer is convolutional if for each node, only a specific region of the previous layer is processed, and these regions can overlap each other.

A layer is of pooling type if for each node, only a specific region of the previous layer is processed, and these regions cannot overlap each other.

Convolutunial layers are used in general when closer nodes in a region are in a way similar and, and it makes sense that one node in the next layer processes just the nodes in the previous layer in a certain region.

Pooling layers, but sometimes convolutional layers as well, are used to reduce the number of weights and biases.

Recurrent neural networks involve cycles that are used to process information obtained at a previous time. This is useful when prediction depends on the previous data. This can happen for instance in a speech(stream of words, where all the words matter) or in a song.

In this work, the choice has fallen on feed forward neural network with dense layers. This choice has been done with some reasons:

- The terms (and neither the genes) are ordered in certain way in the files, so it doesn't make sense to process just a region of the nodes (which are terms) in each layer
- The output of the neural network should have the same shape of the input. So it's wrong to use convolutional or pooling layers because they would reduce the shape of the input
- The terms are not related each other, so using recurrent neural networks is useless

It would have been easier to work with full annotation matrices in the program, but sparse matrices have been used instead. In fact sparse matrices store only the indices of the annotations which are different than 0. It requires a little more work to use sparse matrices, but sparse matrices in this case occupy much less memory. In fact an average number of terms and genes for each annotation matrices is 5,000. This means there are an average of 25,000,000 of annotations for each annotation matrix. For each gene there are an average of only 50 annotations which are 1. This means a sparse matrix memorise just 250,000 values instead of 25,000,000. Beside the used data type for each annotation, this way of memorising the annotation matrix reduces a lot the needed memory.

A neural network with two hidden layer has been implemented. The sigmoid function has been used as activation function because it gives an output between 0 and 1 for each node in the input, and that's what is needed. An annotation is in fact 0 or 1, and the desired output, which is the likelihood should stand between these two values.

Then a two target organisms training has been implemented.

Instead of training the neural network once with just the annotation matrices of a well known organism, two trainings have been done. This involves that the common terms which the two trainings work with are less. They infact have to be in common between three matrices and not two.

Another reason why used terms are less is the next one:

It should be logical that more recent annotation matrices for the same species have at least the same terms as the older matrices. Unfortunately this not happen always in the annotation matrices worked out by bioinformatics. So, to overcome this problem the common terms to train the network with have to be reduced again, considering also the terms in the more recent matrix of the target species, which acts as the set of labels during the training. Common terms are even reduced because it's necessary to consider the IEA matrix as well.

## 6 Technologies

The neural network has be implemented through the library [TensorFlow](#)<sup>[2]</sup>.

The apis for TensorFlow can be used with python or java. In the project the ones for python have been used. Python 3.5.4 has been used for this work because it's the last stable version of python, and using windows os, tensorflow is compatible with python 3.5.x only.

Several modules for python have been used:

- liac-arff: to read and write arff files as matrices, normal or not
- random: to get a random number of genes from a list and to shuffle a list
- numpy: to use data as its data structure were arrays

The species available in this work are:

- *Bos Taurus*
- *Rattus Norvegicus*
- *Homo Sapiens*
- *Mus Musculus*

The species whose annotation matrices have more annotations are *Mus Musculus* and *Homo Sapiens*. Thus the annotation matrices of these species have been used for training.

A virtual machine on a server provided by the DISI[3] of University of Bologna has been used to do heavy computations for training the neural network and evaluating the accuracy. Its 16 gigabytes of ram and its cpu with four cores and 2000 MHz are way much helpful than a standard home pc to provide these kinds of computations.

## 7 Experimental results

The experiments that have been done are based on annotation matrices of four species:

- *Mus Musculus*
  - A matrix of a 2009 study
    - \* 9818 genes
    - \* 3469 terms
  - A matrix of a 2013 study
    - \* 14503 genes
    - \* 7457 terms
  - A IEA matrix of a 2009 study

- \* 16671 genes
- \* 1714 terms
- A IEA matrix of a 2013 study
  - \* 16753 genes
  - \* 1807 terms
- *Rattus Norvegicus*
  - A matrix of a 2009 study
    - \* 11018 genes
    - \* 4540 terms
  - A matrix of a 2013 study
    - \* 12076 genes
    - \* 7351 terms
  - A IEA matrix of a 2009 study
    - \* 14352 genes
    - \* 2877 terms
  - A IEA matrix of a 2013 study
    - \* 15061 genes
    - \* 4944 terms
- *Bos Taurus*
  - A matrix of a 2009 study
    - \* 733 genes
    - \* 792 terms
  - A matrix of a 2013 study
    - \* 2236 genes
    - \* 2285 terms
  - A IEA matrix of a 2009 study

- \* 11561 genes
- \* 2581 terms
- A IEA matrix of a 2013 study
  - \* 5416 genes
  - \* 2919 terms
- *Homo Sapiens*
  - A matrix of a 2009 study
    - \* 10773 genes
    - \* 3506 terms
  - A matrix of a 2013 study
    - \* 13425 genes
    - \* 6345 terms
  - A IEA matrix of a 2013 study
    - \* 16693 genes
    - \* 4640 terms

It comes easy to see that *Mus Musculus* and *Homo Sapiens* are the most studied species, among the ones above. That's natural because *Mus Musculus* is the scientific name for the lab rat, which is studied a lot due to its similarity to the *Homo Sapiens*, which is the scientific name for the human.

Several experiments have been conducted with these given annotation matrices.

Indeed, *Mus Musculus* and *Homo Sapiens* annotation matrices from studies of 2009 and 2013 have been used to train the neural network. As in the work in[1], the M value was fixed on a value of 20, and the tolerance was fixed to an appropriate value of 0.8.

The sigmoid activation function was chosen to be used because



it's not linear and the majority of its outputs is close to 0 or 1, and that should be the output of the neural network in our case. So experiments were conducted varying the number of iterations and/or the batch size.

*Homo Sapiens* and *Mus Musculus* annotation matrices from 2009 and 2013 were used. Varying the batch size and the number of iterations for each training brought these results in the following table.

Table 2: accuracies from *Homo Sapiens* and *Mus Musculus* trainings tested with *Bos Taurus*; M=20

|            | Batch size | 10   | 25   | 50   | 100  |
|------------|------------|------|------|------|------|
| Iterations |            |      |      |      |      |
| 10         |            | 0.46 | 0.45 | 0.46 | 0.45 |
| 100        |            | 0.38 | 0.34 | 0.29 | 0.24 |
| 500        |            | 0.43 | 0.33 | 0.24 | 0.22 |
| 1000       |            | 0.50 | 0.32 | 0.22 | 0.20 |
| 5000       |            | 0.62 | 0.45 | 0.35 | 0.27 |
| 10,000     |            | 0.68 | 0.56 | 0.50 |      |
| 20,000     |            | 0.79 | 0.70 |      |      |
| 30,000     |            | 0.81 |      |      |      |

Then, other possible species for training the model and evaluation of the accuracy have been tested. The batch size of 10 was kept because it showed the best accuracies in the previous test.

Table 3: accuracies from *Rattus Norvegicus* and *Mus Musculus* trainings tested with *Bos Taurus*; M=20

|            | Batch size | 10   |
|------------|------------|------|
| Iterations |            |      |
| 10         |            | 0.44 |
| 100        |            | 0.33 |
| 1000       |            | 0.47 |
| 5000       |            | 0.61 |
| 10,000     |            | 0.69 |

Table 4: accuracies from *Rattus Norvegicus* and *Homo Sapiens* trainings tested with *Bos Taurus*; M=20

|            | Batch size | 10   |
|------------|------------|------|
| Iterations |            |      |
| 10         |            | 0.45 |
| 100        |            | 0.42 |
| 1000       |            | 0.45 |
| 5000       |            | 0.62 |
| 10,000     |            | 0.69 |

Table 5: accuracies from *Mus Musculus* and *Homo Sapiens* trainings tested with *Rattus Norvegicus*; M=20

|            | Batch size | 10   |
|------------|------------|------|
| Iterations |            |      |
| 10         |            | 0.48 |
| 100        |            | 0.50 |
| 1000       |            | 0.55 |
| 5000       |            | 0.71 |
| 10,000     |            | 0.79 |
| 20,000     |            | 0.86 |
| 30,000     |            | 0.87 |

Reaching an accuracy of 0.87 with the training with *Mus Musculus* and *Homo Sapiens*, applied to *Rattus Norvegicus*, other M values have been tried.

Table 6: accuracies from *Mus Musculus* and *Homo Sapiens* trainings tested with *Rattus Norvegicus*; M=5

|            | Batch size | 10   |
|------------|------------|------|
| Iterations |            |      |
| 100        |            | 0.50 |
| 1000       |            | 0.56 |
| 10,000     |            | 0.80 |

Table 7: accuracies from *Mus Musculus* and *Homo Sapiens* trainings tested with *Rattus Norvegicus*; M=10

|            | Batch size | 10   |
|------------|------------|------|
| Iterations |            |      |
| 100        |            | 0.49 |
| 1000       |            | 0.51 |
| 10,000     |            | 0.79 |

Table 8: accuracies from *Mus Musculus* and *Homo Sapiens* trainings tested with *Rattus Norvegicus*; M=40

|            | Batch size | 10   |
|------------|------------|------|
| Iterations |            |      |
| 100        |            | 0.44 |
| 1000       |            | 0.48 |
| 10,000     |            | 0.78 |

Table 9: accuracies from *Mus Musculus* and *Homo Sapiens* trainings tested with *Rattus Norvegicus*; M=100

| Iterations | Batch size 10 |
|------------|---------------|
| 100        | 0.41          |
| 1000       | 0.41          |
| 10,000     | 0.63          |

## 8 Conclusions

As it's easily possible to see, among the tried batch sizes, the batch size that obtained the best accuracy is 10. Then using a batch size of 10, the accuracy generally grows as the number the iterations increase. Indeed, the best obtained accuracy is 0.87 with a batch size of 10 and 30000 iterations. A reason why the accuracy results lower with bigger batch size could be an over-fitting of the network.

As the results show, the best obtained accuracy, which is 0.87, was obtained training the network with *Mus Musculus* and *Homo Sapiens*, and then applying the network to *Rattus Norvegicus*. These are better results than the ones obtained obtained testing the same way trained network with *Bos Taurus*. This may be influenced by the similarity between the species of *Mus Musculus* and *Rattus Norvegicus*, and *Homo Sapiens* and *Rattus Norvegicus*. Training the network with two species and not one could have influenced the model, and may has given the model better results for an average testing, i.e. testing for species that are not necessarily similar to the ones used for training. In-fact, training with just one species may makes the network too specific for the species it is trained with.

The experimental results also show that generally the M value shouldn't be greater than 20. Testing the accuracies with M values lower or equal than 20, the results are pretty similar. Testing

the accuracies with M values of 40 and 100, they get lower.

It's possible that training the network with annotation matrices of the same species as the testing genes could influence in a better way the accuracy, but using a cross-organism training makes more sense and it's more usable by bioinformatics. Training and testing the model just with annotations for a limited amount of terms is for sure bad, because there are some terms which are specific for some species, and their annotations would be unpredictable through this methods.

The architecture of the model then, could have influenced in a strong way the speed of the whole process of training and testing, and also the accuracy. Three layers have been infact used, using the sigmoid function as the activation function. A problem network suffer is infact a cold start up. Training the network is costly and requires a huge amount of time(several days, depending on server cpu and memory, and on number of iterations and batch size), that has to be used each time, for each testing. Indeed, server capabilities and a costly method have strongly influenced the proposed results, which could be more.

Finally, results show accuracy grows with increasing iterations; so at the moment it seems that vanishing gradient didn't happen yet, even though it could happen increasing them more. Sigmoid function is giving an output bewteen 0 and 1, so it is subject to vanishing gradient problem. However, in this work it wouldn't changed anything to use another activation function, by this point of view, because the outputs of the model are by default included between 0 and 1.

At the moment this results are not excellent and aren't usable in a professional context, even though an accuracy of 0.87 is a good step to begin other researches. However, this method can be used for validating previously predicted annotations.

## 9 Future developments

There are some shrewdness that could be done in order to improve the accuracy or the flexibility. First the method with many more matrices from several species and above all, from different times of studies, could give information on patterns to choose the best training matrices, even from different species. Then, it would be possible to use this method for protein annotations, or even for binary, and eventually sparse, matrices composed by ones and zeros, even from a different subject of studies. Flexibility could be reached with a more static neural network that expects all the times the same number of terms. In this way, weights and biases could be saved into a file and then just used for each testing, without training the network each time.

Having predicted just a part of possible annotations for a gene, it could be useful to build even a machine learning method to establish a pattern among annotations of different terms, and so, once predicted the annotations for the common terms, even predict the missing ones through this method.

Further, several experiments on different species, and not just four, could be done in order to find the species which train the network best. Moreover, several experiments on the  $M$  value could be done, getting the right value of  $M$ , understanding how this value influences the final accuracy.

A completely different approach for the architecture of the method could be used then. Instead of choosing the genes with at least  $M$  true annotations, a greater weight could be assigned to genes with more true annotations.

Then working just with common terms hinders much flexibility. It impedes to use always the same weights and biases to test the network. Finding a way to let dimensions of matrices not influence the shape of the network would be great, even though it could bring a loss of accuracy.

Finally, as the experimental results may suggest, similarity be-

tween different species could influence final accuracy. Thus, finding best species to train the network in order to find the best accuracies on the target species, could improve the accuracy.

## 10 Appendix

### 10.1 Implementation of the software

The software has been implemented using python 3.5. The basic steps for the working of the software, which build, trains, and test a neural network are the following ones:

- Get from the input the annotation matrices to work with(train,test).
- Get the parameters(M, batch size, number of iterations).
- build the structure of the neural network, basing on the common terms, each layer infact has as many nodes as the number of common terms.
- train the neural network with the annotations that refer to the common terms previously got, and that refer to genes which have at least M positive annotations among the one which refer to common terms.
- test the neural network with an annotation matrix(not used for the training)
- compare the obtained annotations to the ones of the actual annotation matrix they should correspond to. Compute the accuracy

Here a schematic part of the code used for this work

```
1 import tensorflow as tf
2 import arff
3
```

```

4 import numpy as np
5 import random
6 import copy
7 import sys
8
9 #getting from command line(sys) all the parameters
10 ancestorList = getAncestorList(pathAncestor)
11 annotationsTargetOld = getAnnotationMatrix(pathBold)
12 annotationsSourceNew = getAnnotationMatrix(pathAnew)
13 annotationsSourceOld = getAnnotationMatrix(pathAold)
14 annotationsTargetNew = getAnnotationMatrix(pathBnew)
15 annotationsTargetNewIea = getAnnotationMatrix(pathBnewIea
)
16 annotationsSourceNew2 = getAnnotationMatrix(pathAnew2)
17 annotationsSourceOld2 = getAnnotationMatrix(pathAold2)
18 termsA = getTermsAnnotationMatrix(annotationsSourceOld)
19 termsB = getTermsAnnotationMatrix(annotationsTargetOld)
20 termsAnew = getTermsAnnotationMatrix(annotationsSourceNew
)
21 termsBnew = getTermsAnnotationMatrix(annotationsTargetNew
)
22 termsBnewIea = getTermsAnnotationMatrix(
annotationsTargetNewIea)
23 termsA2 = getTermsAnnotationMatrix(annotationsSourceOld2)
24 termsA2new = getTermsAnnotationMatrix(
annotationsSourceNew2)
25
26 #getting common terms
27 commonTerms = getCommonTerms(termsA , termsB , termsAnew ,
termsBnew , termsA2new , termsA2 , termsBnewIea)
28 termlength = getTermsLength(commonTerms)
29
30
31 #building the network
32 x = tf.placeholder(tf.float32 , [None, termlength])
33
34 W1 = tf.Variable(tf.random_normal([ termlength , termlength
], stddev=0.35))
35 b1 = tf.Variable(tf.zeros([ termlength ]))
36 y1 = tf.sigmoid(tf.matmul(x, W1) + b1) # predicted
annotations
37
38 W2 = tf.Variable(tf.random_normal([ termlength , termlength
], stddev=0.35))
39 b2 = tf.Variable(tf.zeros([ termlength ]))
40 y2 = tf.sigmoid(tf.matmul(y1, W2) + b2)
41
42 W3 = tf.Variable(tf.random_normal([ termlength , termlength
], stddev=0.35))

```



```

43 b3 = tf.Variable(tf.zeros([ termlength ]))
44 y3 = tf.sigmoid(tf.matmul(y2, W3) + b3)
45
46 y = y3
47
48 y_ = tf.placeholder(tf.float32 , [None, termlength])
49 cross_entropy = tf.reduce_mean(tf.nn.
softmax_cross_entropy_with_logits(logits=y, labels=y_))#
error function
50 train_step = tf.train.GradientDescentOptimizer(0.05).
minimize(cross_entropy)
51
52 #getting training data
53 genesList = getGenesList(commonTerms,
annotationsSourceOld , annotationsSourceNew)
54 trainingList = getTrainingList(genesList ,
annotationsSourceOld , annotationsSourceNew , batchsize ,
commonTerms)
55
56 #training the neural network
57 for i in range(iterations):
58     trainingList = sess.run(tf.random_shuffle(
trainingList))
59     batch_xs = trainingList[0]
60     batch_ys = trainingList[1]
61     arr_xs = np.asarray(batch_xs)
62     arr_ys = np.asarray(batch_ys)
63     sess.run(train_step , feed_dict={x: arr_xs ,y_: arr_ys
})
64
65 '''then repeat the same for the other training matrices
,,,
66
67 annotationsTargetOldannots = []
68 annotationsTargetNewannots = []
69 annotationsTargetNewannotsIea = []
70 commonGenes = getCommonGenes(annotationsTargetOld ,
annotationsTargetNew , annotationsTargetNewIea)
71 for gene in commonGenes:
72     annotationsTargetOldannots.append(getAnnotations(gene
, annotationsTargetOld , commonTerms))
73     annotationsTargetNewannots.append(getAnnotations(gene
, annotationsTargetNew , commonTerms))
74     annotationsTargetNewannotsIea.append(getAnnotations(
gene , annotationsTargetNewIea , commonTerms))
75
76     sigm = sess.run(y, feed_dict={x: np.asarray(
annotationsTargetOldannots)})#evaluating predicted
annotations

```

```

77     sigm=correctLikelihoodMatrix(sigm , ancestorList ,
commonTerms)
78     prec=evaluateAccuracy(sigm , annotationsTargetNewannots ,
annotationsTargetNewannotsIea , tolerance)#evaluating
accuracy

```

## 10.2 Usage of the software

The program has a terminal interface that allows the user to insert parameters. These parameters are:

- Paths of the arff files for the used annotation matrices
- Path of the file where to write the predicted annotation matrix
- Path of the file where to write the obtained accuracy
- M value. M is the minimum number of true annotations a gene has to have to be considered for the training
- batch size of the testing data. It means the number of genes used for training
- number of iterations per training
- Tolerance value used to evaluate accuracy

Anyway, these information are described better if calling the python script with a "-help" option.

The program trains the network, and saves the predicted annotation matrix and accuracy to two different files. In the saved arff file, the predicted annotations which are greater than 0.5 are saved as one, the other ones are saved as 0.

It would have been easier for the user, to apply the input, an annotation matrix, to a previously built neural network, without training the network each time. Unfortunately this was unavoidable because neural network have a fixed number of weights and

biases that depend on the terms and genes of the training matrices and of the input matrix(the one used to get prediction and accuracy) as well.

## References

- [1] Giacomo Domeniconi, Marco Masseroli, Gianluca Moro, and Pietro Pinoli. Cross-organism learning method to discover new gene functionalities. *Computer methods and programs in biomedicine*, 126:20–34, 2016.
- [2] TensorFlow. Library for neural networks. <https://www.tensorflow.org/>
- [3] DISI, university of Bologna, dipartimento di informatica-scienze e ingegneria(department of informatics- science and engineering) <http://www.informatica.unibo.it/it>
- [4] Practical Recommendations for Gradient-Based Training of Deep Architectures. Yoshua Bengio. Version 2, Sept. 16th, 2012.
- [5] Theory of the Backpropagation Neural Network Robert Hecht-Nielsen HNC, Inc. 5501 Oberlin Drive San Diego, CA 92121 619-546-8877 and Department of Electrical and Computer Engineering University of California at San Diego La Jolla, CA 92139
- [6] P. Khatri, B. Done, A. Rao, A. Done, S. Draghici, A semantic analysis of the annotations of the human genome, *Bioinformatics* 21 (16) (2005) 3416–3421.
- [7] B. Done, P. Khatri, A. Done, S. Draghici, Predicting novel human gene ontology annotations using semantic analysis, *IEEE/ACM Trans Comput Biol Bioinform* 7 (1) (2010) 91–99.

- [8] M. Masseroli, M. Tagliasacchi, D. Chicco, Semantically improved genome-wide prediction of Gene Ontology annotations, in: Proc IEEE Int Conf Intell Syst Design App (ISDA 2011), IEEE, 2011, pp. 1080–1085.
- [9] B. Done, P. Khatri, A. Done, S. Draghici, Predicting novel human gene ontology annotations using semantic analysis, IEEE/ACM Trans Comput Biol Bioinform 7 (1) (2010) 91–99.
- [10] B. Done, P. Khatri, A. Done, S. Draghici, Semantic analysis of genome annotations using weighting schemes, in: Proc IEEE Symp Comput Intell Bioinforma Comput Biol (CIBCB 2007), IET, 2007, pp. 212–218.
- [11] P. Pinoli, D. Chicco, M. Masseroli, Weighting scheme methods for enhanced genomic annotation prediction, in: Computational Intelligence Methods for Bioinformatics and Biostatistics. LNCS (Lecture Notes in Bioinformatics), Vol. 8452, Springer, 2014, pp. 76–89.
- [12] S. T. Dumais, G. W. Furnas, T. K. Landauer, S. Deerwester, R. Harshman, Using latent semantic analysis to improve access to textual information, in: Proc ACM SIGCHI Conf Hum Factors Comput Syst, ACM, 1988, pp. 281–285.
- [13] T. Hofmann, Probabilistic latent semantic indexing, in: Proc Int ACM SIGIR Conf Res Dev Inf Retr (RDIR 1999), ACM, 1999, pp. 50–57.
- [14] M. Masseroli, D. Chicco, P. Pinoli, Probabilistic Latent Semantic Analysis for prediction of Gene Ontology annotations, in: Proc Int Joint Conf Neural Netw (IJCNN 2012), IEEE, 2012, pp. 2891–2898.
- [15] P. Pinoli, D. Chicco, M. Masseroli, Enhanced probabilistic latent semantic analysis with weighting schemes to predict

- genomic annotations, in: Proc IEEE Int Conf Bioinformatics Bioeng (BIBE 2013), IEEE, 2013, 92, pp. 1–4.
- [16] P. Khatri, B. Done, A. Rao, A. Done, S. Draghici, A semantic analysis of the annotations of the human genome, *Bioinformatics* 21 (16) (2005) 3416–3421.
- [17] D. M. Blei, A. Y. Ng, M. I. Jordan, Latent dirichlet allocation, *J Mach Learn Res* 3 (2003) 993–1022.
- [18] A. Perina, P. Lovato, V. Murino, M. Bicego, Biologically-aware latent dirichlet allocation (balda) for the classification of expression microarray, in: *Pattern Recognition in Bioinformatics*, Springer, 2010, pp. 230–241.
- [19] M. Bicego, P. Lovato, B. Oliboni, A. Perina, Expression microarray classification using topic models, in: *Proc ACM Symp Appl Comput*, ACM, 2010, pp. 1516–1520.
- [20] P. Pinoli, D. Chicco, M. Masseroli, Latent Dirichlet Allocation based on Gibbs Sampling for gene function prediction, in: *Proc IEEE Symp Comput Intell Bioinforma Comput Biol (CIBCB 2014)*, IEEE, 2014, pp. 1–8.
- [21] T. Griffiths, Gibbs sampling in the generative model of Latent Dirichlet Allocation, *Stanford University* 518 (11) (2002) 1–3.
- [22] G. Casella, E. I. George, Explaining the Gibbs sampler, *Am Stat* 46 (3) (1992) 167–174.
- [23] I. Porteous, D. Newman, A. Ihler, A. Asuncion, P. Smyth, M. Welling, Fast collapsed Gibbs sampling for latent Dirichlet allocation, in: *Proc ACM SIGKDD Int Conf Knowl Discov Data Min (KDDM 2008)*, ACM, 2008, pp. 569–577.
- [24] D. Warde-Farley, S. L. Donaldson, O. Comes, K. Zuberi, R. Badrawi, P. Chao, M. Franz, C. Grouios, F. Kazi, C. T.

- Lopes, et al., The Gene- MANIA prediction server: biological network integration for gene prioritization and predicting gene function, *Nucleic Acids Res* 38 (Web Server issue) (2010) W214–W220.
- [25] X. Li, Z. Zhang, H. Chen, J. Li, Graph kernel-based learning for gene function prediction from gene interaction network, in: *Proc IEEE Int Conf Bioinformatics Biomed (BIBM 2007)*, IEEE, 2007, pp. 368–373.
- [26] H. Mi, A. Muruganujan, P. D. Thomas, PANTHER in 2013: modeling the evolution of gene function, and other gene attributes, in the context of phylogenetic trees, *Nucleic Acids Res* 41 (Database issue) (2013) D377– D386.
- [27] X. Deng, H. Ali, A hidden markov model for gene function prediction from sequential expression data, in: *Proc IEEE Comput Sys Bioinform Conf*, IEEE, 2004, pp. 670–671.
- [28] O. D. King, R. E. Foulger, S. S. Dwight, J. V. White, F. P. Roth, Predicting gene function from patterns of annotation, *Genome Res* 13 (5) (2003) 896–904.
- [29] Y. Tao, L. Sam, J. Li, C. Friedman, Y. A. Lussier, Information theory applied to the sparse Gene Ontology annotation network to predict novel gene function, *Bioinformatics* 23 (13) (2007) 529–538.
- [30] F. Minneci, D. Piovesan, D. Cozzetto, D. T. Jones, FFPred 2.0: Improved homology-independent prediction of Gene Ontology terms for eukaryotic protein sequences, *PloS One* 8 (5) (2013) e63754.
- [31] N. Mitsakakis, Z. Razak, M. D. Escobar, J. T. Westwood, Prediction of *Drosophila melanogaster* gene function using Support Vector Machines, *BioData Min* 6 (1) (2013) 8.

- [32] L. Cheng, H. Lin, Y. Hu, J. Wang, Z. Yang, Gene function prediction based on the Gene Ontology hierarchical structure, *PloS One* 9 (9) (2014) e107187.
- [33] D. Stojanova, M. Ceci, D. Malerba, S. Dzeroski, Using PPI network autocorrelation in hierarchical multi-label classification trees for gene function prediction, *BMC Bioinformatics* 14 (2013) 285.
- [34] O. G. Troyanskaya, K. Dolinski, A. B. Owen, R. B. Altman, D. Botstein, A Bayesian framework for combining heterogeneous data sources for gene function prediction (in *Saccharomyces cerevisiae*), *Proc Natl Acad Sci USA* 100 (14) (2003) 8348–8353.
- [35] Z. Barutcuoglu, R. E. Schapire, O. G. Troyanskaya, Hierarchical multi-label prediction of gene function, *Bioinformatics* 22 (7) (2006) 830–836.
- [36] A. J. Pérez, C. Perez-Iratxeta, P. Bork, G. Thode, M. A. Andrade, Gene annotation from scientific literature using mappings between keyword systems, *Bioinformatics* 20 (13) (2004) 2084–2091.
- [37] S. Raychaudhuri, J. T. Chang, P. D. Sutphin, R. B. Altman, Associating genes with gene ontology codes using a maximum entropy analysis of biomedical literature, *Genome Res* 12 (1) (2002) 203–214.
- [38] G. Pandey, V. Kumar, M. Steinbach, Computational approaches for protein function prediction: A survey, Tech. Rep. TR 06-028, Department of Computer Science and Engineering, University of Minnesota, Minneapolis, MN, USA (2006).
- [39] A. K. Tiwari, R. Srivastava, A survey of computational intelligence techniques in protein function prediction, *Int J Proteomics* 2014 (2014) 845479.

- [40] M. Zitnik, B. Zupan, Matrix factorization-based data fusion for gene function prediction in bakers yeast and slime mold, in: *Pac Symp Biocomput*, World Scientific, 2014, pp. 400–411.
- [41] M. A. Huynen, B. Snel, V. van Noort, Comparative genomics for reliable protein-function prediction from genomic data, *Trends Genet* 20 (8) (2004) 340–344.
- [42] D. Chicco, M. Tagliasacchi, and M. Masseroli, "Genomic annotation prediction based on integrated information". *Computational Intelligence Methods for Bioinformatics and Biostatistics*, Springer Berlin Heidelberg, pp. 238-252, 2012.
- [43] Emre Sefer and Carl Kingsford. Metric Labeling and Semi-metric Embedding for Protein Annotation Prediction. In *Research in Computational Molecular Biology*, 2011.