UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

CORSO DI LAUREA MAGISTRALE IN BIOINGEGNERIA DELLA RIABILITAZIONE

Enhancing protein function prediction: integrating neural networks with Argot 2.5 web server

Relatore: Prof. Enrico Lavezzo

Laureando: Gabriele Gradara

Correlatori: Prof. Stefano Toppo,
Dott. Emilio Ispano

ANNO ACCADEMICO 2023 – 2024

Data di laurea 10/7/2024

Contents

Abstract

The prediction of protein function through in silico methods is becoming more prevalent and important. An increasing amount of gene products is continuously sequenced thanks to the accessibility of New Generation Sequencing (NGS) techniques. Unfortunately, this high output of information is not met by an appropriate amount of research regarding the functions of those proteins. Due to the accurate but resource-intensive nature of experimental evidence, the field of molecular biology cannot keep up with the number of gene products that are discovered regularly.

The category of Automated Function Prediction (AFP) techniques was born to address this problem and generate the required annotations necessary to describe the functions of the proteins that cannot be adequately studied due to time or resource constraints. AFPs aim to be a fast and cost-effective substitute for wet lab research while keeping a similar level of accuracy.

Argot 2.5 is an AFP created by the group of Computational Medicine of the University of Padua, designed to predict protein function starting from its amino-acidic sequence. The efforts of this project revolve around the implementation of a new evaluation system that estimates the accuracy of each prediction through the use of a neural network. This neural network will weigh Argot's multiple outputs to provide to the end user a curated selection of the most reliable functions predicted by Argot.

In addition, the research team noticed that Argot predicted some functions that were wrong at the time of the prediction but were recognized as right years later after the publication of further experimental evidence. This means that Argot may have predictive capabilities that can be used to lead the direction of the experimental research. This project also comprehends the development of a second neural network focused on finding which function predictions, scored as false positives at the time they were generated, have a high probability of being corroborated in the future, which may be worthy of further focused research.

Both neural networks developed in this thesis project provided an increase in the performance of the function predictions generated by the Argot 2.5 server and will be implemented in the novel version of the tool.

Chapter 1

Introduction

1.1 Gene ontology

With the advent of great projects in the field of molecular biology, aiming to sequence the complete genome of eukaryotic life forms, like the budding yeast [1] (completed in 1996), the nematode worm [2] (completed in 1998) and later the Human Genome Project [3] (started in 1990), the necessity for a systematic approach to data categorization and organization became apparent. This problem was exacerbated by the vast amount of data obtained via the New Generation Sequencing (NGS) techniques, which drastically increased the output of protein sequences. The NGS techniques also increased the involvement of computers and software in the field, eliciting the requirement of a machine-readable implementation.

Due to the absence of an authority, in the past, databases in the field of molecular biology were highly heterogeneous. They had different scopes, organizations and even different naming conventions for the same concepts [4]. To try to curb this problem and unify and standardize the structure and contents of the databases, the Gene Ontology project was born.

The Gene Ontology (GO) project is a collaborative effort to provide structured and controlled vocabularies and classifications that cover several domains of molecular and cellular biology [5]. This project aims to develop a set of controlled structured vocabularies known as ontologies[1]. The resources and software required for the annotations of genes, gene products and sequences are publicly available.

Each Gene Ontology Annotation (or GOA) describes a particular attribute of gene products (an umbrella term that consists of DNA segments, proteins, etc.) in three different domains:

- Molecular function (MF): describes activities at the molecular level. Where, when or in what context this activity is performed is not specified.
- Biological process (BP): describes larger processes, accomplished by one or more ordered assemblies of molecular functions. High-level processes may have sub-types that allow to specify the process in detail.
- Cellular component (CC): describes locations, at the levels of sub-cellular structures and macromolecular complexes.

[1]In this context an ontology is a collection of entities and definitions useful to explain molecular biology phenomena. An ontology correlates hierarchically these concepts with "is in a subset of" and "is a member of" relations [4]. An example can be seen in Figure 1.1.

Every domain has its own ontology. Each protein may have GO annotations belonging to multiple domains, usually from all three [5]. All the GO terms (or classes) have a name, a unique seven-digit ID prefaced by GO: (e.g., GO:1904659 for the glucose trans-membrane transport), and a brief description of their definition.

The graph of an ontology is a directed acyclic graph (or DAG). Each child term (the ones closer to the leaves) points to its parents (the ones closer to the roots), a child can have multiple parents and vice versa. The parent terms are more general in scope while the children are more specific. A general annotation may branch out to more specific processes or properties that are more precisely defined [6].

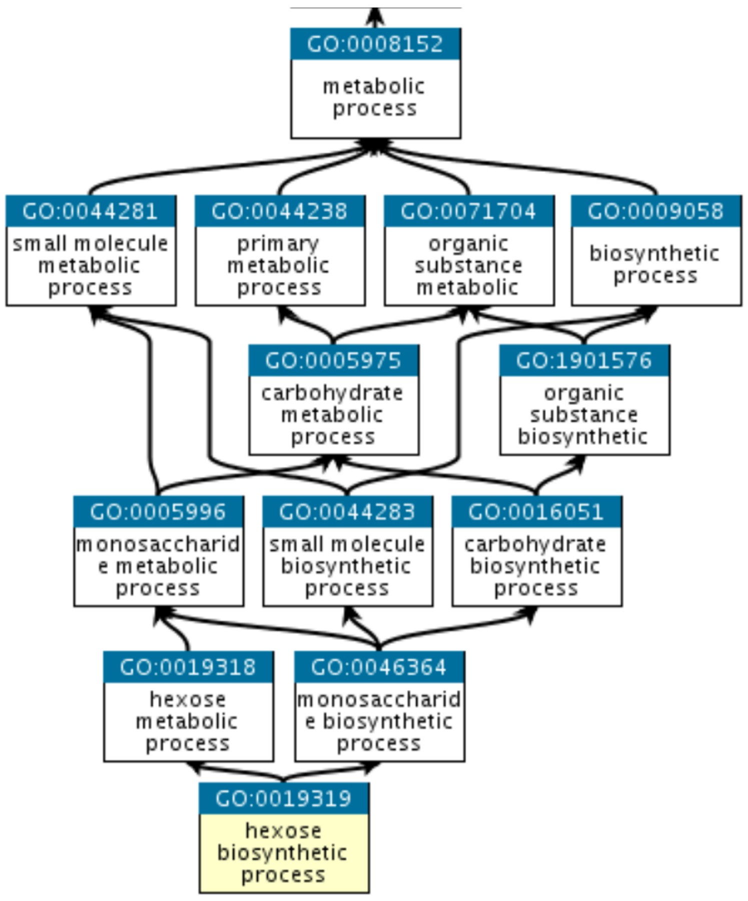

GO:0008152
metabolic process
GO:0044281
small molecule metabolic process
GO:0044238
primary metabolic process
GO:0071704
organic substance metabolic
GO:0009058
biosynthetic process
GO:0005975
carbohydrate metabolic process
GO:1901576
organic substance biosynthetic
GO:0005996
monosaccharide metabolic process
GO:0044283
small molecule biosynthetic process
GO:0016051
carbohydrate biosynthetic process
GO:0019318
hexose metabolic process
GO:0046364
monosaccharide biosynthetic process
GO:0019319
hexose biosynthetic process

Figure 1.1: Representation of a small piece of the GO graph. The first layers of the branch have more general biological concepts while the bottom ones are more specific [6].

1.2 The annotation problem

Due to the widespread use of NGS techniques, we have access to an abundance of protein sequences like never before, however, the information regarding the functions conducted by those gene products lags behind [7]. The amount of resources required to collect experimental evidence to learn what each protein does is disproportionally higher than the resources required to acquire the sequence in the first place. To bridge this gap research efforts have been made to develop automated protein function prediction (AFP) tools that could compute these functions without the costly and time-consuming task of leading experiments.

There are two main approaches to AFP development: the deterministic approach and the machine learning (ML) approach. The deterministic approach uses sequence-based and structure-based methods for their function predictions. The sequence-based method relies on the similarity between the amino acidic sequence of the gene product to be annotated and the sequences with known functions available in databases such as UniProt [8], [9]. The structure-based method works upon the principle that proteins with a similar tridimensional shape have similar functions. This approach can circumvent problems brought by sequences that do not share a high degree of similarity. An example of structure-based AFP is the FFPred3 [10] which uses a collection of several support vector machines (SVM), each one tasked to assign a specific GO term to the considered gene product.

The combined-based method is a multi-source method that uses a combination of deterministic approaches like sequence-based and structural-based methods. The exploitation of multiple sources of information gives an edge in performance to these AFPs, but they are still unable to outperform manual and experimental annotations [8]. Furthermore, the high complexity caused by the combination of multiple sources of information for a single prediction reduces the applicability of these methods.

The ML-based approaches see function prediction as a large-scale multilabel classification problem, this problem can be solved via a classifier model, like a neural network, in the case of the purely ML-based approach or via the use of embeddings for the embeddings-based methods. These innovative approaches are promising, the state-of-the-art right now in protein function prediction is DeepGoPlus [11], an embedded-based AFP that is entirely data-driven (does not require manually crafted inputs) [8].

To evaluate each tool the CAFA [7] (Critical Assessment of Functional Annotation) community challenge has been created. In each CAFA challenge each AFP research team submits its tool and then, after the deadline, the researchers leading the challenge collect novel gene product annotations via published literature, or experiments conducted by themselves. The use of new annotations guarantees the unbiasedness of the dataset used for testing, however, a lot of time is required to collect a satisfactory amount of data, so the results of the challenge are often delayed by years.

1.3 Argot 2.5

Argot 2.5 (Annotation Retrieval of Gene Ontology Terms) is a web server designed to predict protein function [12]. Its algorithm allows for an automated prediction of the GO terms of given DNA or protein sequences [13]. Argot 2.5 capitalizes on the similarities between the input sequences and the ones present in the databases that are already characterized, this means that these gene products were already annotated with GO terms.

1.3.1 The algorithm

Argot 2.5 uses the UniProt data bank [9] and keeps only the proteins with GO annotations. Then these proteins are clustered at 90% sequence identity and 80% overlap with the longest sequence of the cluster using CD-HIT [14]. Only the representatives for each cluster, called seeds, are used as sequence space for BLAST searches [15], greatly improving computational times and reducing the over-representation of almost identical sequences that saturate the BLAST hits list. To avoid the loss of information each seed inherits the GO annotations of its cluster members. An HMMER3[2] [16] search against Pfam[3] [17] is also performed. To quantify the quality of the hits a metric called weights (W) is produced, the weights are proportional to the reliability of the hit (higher is better). To further increase performance and computation time a set of taxonomical constraints provided by the Gene Ontology Consortium (GOC) [18] and the rules generated by the Functional Taxonomy Information System (FunTaxIS) [19] are implemented. These constraints are then used to delete or replace, with a suitable substitute, GO terms that do not belong to the species from which the gene products come. The results retrieved from these steps are used to query the GOA databank: the collected GO terms are ranked according to both the significance of the hit they come from (provided by the e-value of the BLAST search) and their occurrence in the results. Then, all the paths between the input GO and the root are computed, and all the excess GOs are excluded, creating the GO-slim. These GOs are grouped in sets according to their semantic similarity, this way the nodes that share a strong biological relationship form a unique informative group, and only the most specific and high-scoring annotations are considered. To quantify semantic similarity the simgic metric is used, and is implemented with the following formula:

$$simgic(GO_A, GO_B) = \frac{\sum_{t \in prop(GO_A \cap prop(GO_B)} IC(t)}{\sum_{t \in prop(GO_A \cup prop(GO_B)} IC(t)}$$

where GO_A and GO_B are two GO terms and their propagation up to the root is given by $prop(GO_A)$ and $prop(GO_B)$. Simgic is defined as the sum of the Information Content (IC) of each term t in the intersection of $prop(GO_A)$ and $prop(GO_B)$ divided by the sum of the IC of each term t in their union. IC has been calculated for each GO term according to Resnik [20]:

$$IC(t) = -log[p(t)]$$

In which $p(t)$ is the probability of usage of the term in the corpus, which in our case corresponds to the sum of the occurrences of the term and its descendants in the GOA database.

To filter isolated GO terms and rank the remaining ones, three other metrics have been devised. The first one, the Group Score (GrS) is the sum of the cumulative Internal Confidence InC of the nodes g_j belonging to the k^{th} group Gr_k.

$$GrS(k) = \sum_{\{j:g_j \in Gr_k\}} InC(g_j)$$

The Internal Confidence InC is a cumulative measure that takes into account the global cumulative weight distributions defined as $W(g) = \sum\limits_{\{j:g \mapsto g_j\}} w_j$ that is the

[2]A search algorithm that allows the use of probabilistic inference [16].

[3]A widely used database of protein families [17].

sum of the weight w of a GO term g plus the weights of its children, and the sum of the cumulative weight of the root node:

$$InC(g_i) = \frac{\sum\limits_{\{j:\ g_i \mapsto g_j\}} w_j}{\sum\limits_{\{j:\ g_{root} \mapsto g_j\}} w_j} = \frac{W(g_i)}{W(g_{root})} \tag{1.1}$$

The second score is called Z score and is calculated for each extracted GO term g_i as follows:

$$Z(g_i) = \frac{W(g_i) - \overline{W}}{\sigma}$$

where $\overline{W}$ is the weight of the root node divided by the total number of the retrieved GO nodes, while σ is the standard deviation of all the weights. Z-score helps us determine the significance of a retrieved GO via the BLAST and HMMER3 searches compared to the root of the tree, allowing us to prune useless branches.

If the Z-score and the Group Score are below a certain threshold, the corresponding GO terms are discarded. These filtering steps reward those paths, up to the root, that are statistically significant discarding the branches of the GO graph containing nodes with low weights (see in Figure 1.2-ii the discarded path and group). After the filtering phases, the algorithm assigns the third score, the total score TS, to each culled GO term g_i, according to the following formula:

$$TS(g_i) = IC(g_i) \cdot InC^{nc}(g_i) \cdot \frac{InC^{nc}(g_i)}{GrS^{nc}(g_i)} \cdot w_i$$

where InC^{nc} is the non-cumulative internal confidence, calculated as:

$$InC^{nc}(g_i) = \frac{w_i}{\sum\limits_{\{j:\ g_{root} \mapsto g_j\}} w_j} = \frac{w_i}{W(g_{root})} \tag{1.2}$$

Differently from the cumulative Internal Confidence InC defined by (Eq. 1.1), it estimates the local non-cumulative weight distribution, which considers only the weight of the term under analysis. The function GrS^{nc} is the non-cumulative Group Score associated with the k^{th} group Gr_k. It is calculated as the sum of the non-cumulative Internal Confidence InC^{nc} (Eq. 1.2) of the nodes belonging to that group:

$$GrS^{nc}(k) = \sum_{\{g_j \in Gr_k\}} InC^{nc}(g_j)$$

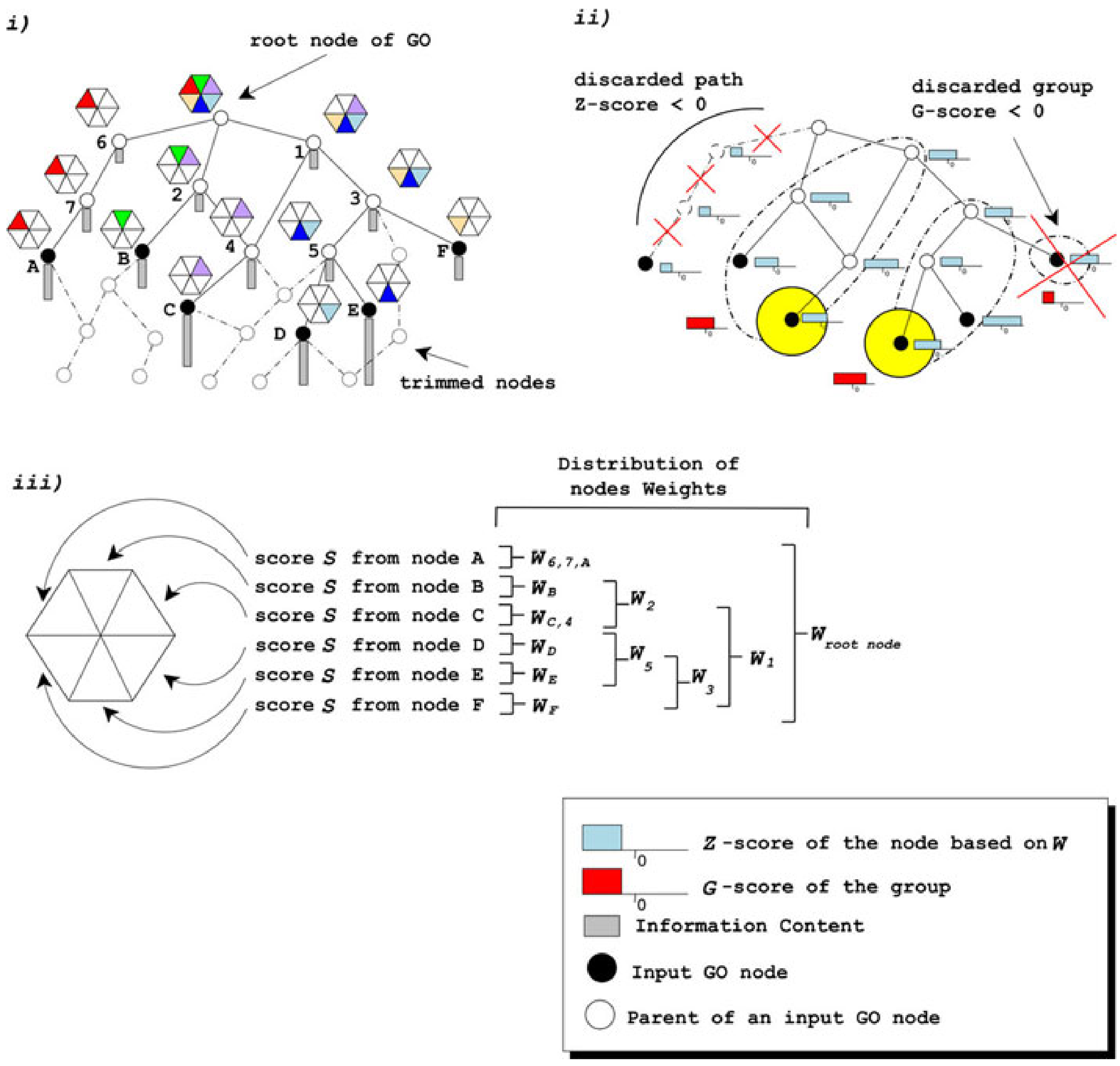


Figure 1.2: Representation of the Argot algorithm. i) The black dots are the GO retrieved by the search and are represented with their weights, while the white dots connected with dashed lines are the trimmed GO terms removed from the GO-slim. ii) The GOs are selected with the use of the Z-score and the G-score. The yellow circles represent the groups with the higher total score (TS) and they will be chosen for the annotation. iii) A representation of how the cumulative weights are calculated for each separate GO node. E.g. W_2 is calculated by the sum of W_B and W_4, but W_4 is equal to W_C because its a reconstructed point from node C, so it does not contribute to the cumulative score [13].

The GO terms with TS above a chosen threshold are extracted and reported. The score rewards those hits that are particularly significant and specific, thanks to the contribution of the Information Content (see yellow circles in Figure 1.2-ii). The non-cumulative measures InC_{nc} and GrS_{nc} have been introduced to guarantee that no biases are introduced due to the scores of child nodes.

1.3.2 Outputs

Argot outputs a table. In this table, each row is a GO term prediction for a specific gene product. These are the columns of that table, and their meaning:

- SeqID: The unique ID for the protein sequence considered.
- GOID: The GO prediction that Argot made for the SeqID. Argot usually makes multiple predictions for each given SeqID.
- Ontology: Which ontological category does the GOID belong to. There are three possible categories: molecular functions, biological processes and cellular components.
- Information content: a numerical quantification of the information provided by each GO. It is related to the specificity of the GO's definition and its rarity.
- Internal confidence: is a metric of reliability of the BLAST search. A higher internal confidence represents a higher compatibility between the evaluated protein sequence and the data bank.
- Total score: The overall reliability of the prediction.
- Group score: This is the sum of the Internal Confidence of all the nodes in a group.

1.3.3 Performance

To evaluate its performance Argot 2.5 joined the CAFA3 [7] challenge. In this challenge, researchers from the worldwide community can submit their algorithm to be independently evaluated and advance the state of the art in protein function prediction. To evaluate performance the team used the F_{max} metric [21]. In this metric $precision(\tau)$ and $recall(\tau)$ represent, respectively, the precision and recall obtained from the predictions that score more than τ itself. τ is chosen to provide the highest possible value, hence the name F_{max}.

$$F_{max} = \max_{\tau} \left\{ \frac{2 \cdot precision(\tau) \cdot recall(\tau)}{precision(\tau) + recall(\tau)} \right\} \tag{1.3}$$

To conduct the challenge, the CAFA team provided novel experimental annotations to use as a benchmark for the submitted algorithms. Considering the no knowledge benchmark, a benchmark containing only proteins with no prior experimental annotations, Argot 2.5 achieved the following results:

Molecular Functions	Biological Processes	Cellular Components
$F_{max} = 0.52$	$F_{max} = 0.38$	$F_{max} = 0.59$

Table 1.1: The results achieved by Argot 2.5 during the CAFA3 challenge.

Even if sequence-based methods such as Argot 2.5 are not innovative from a technical point of view, they can achieve respectable performances, although a machine learning approach appears more promising [8]. Still, comparing directly these algorithms is very challenging due to the tailor-made nature of the datasets used for training and testing purposes. This is a problem that is exacerbated for algorithms that use a ML-approach, due to the fact the composition of these databases will influence their predictions.

1.4 Introduction to neural networks

A neural network (NN) is a biologically inspired computational model, which consists of processing elements (called neurons), and connections between them with coefficients (weights) bound to the connections [22].

The biological neuron and the artificial neuron share some stunning similarities, both in shape and function. Both sum all the inputs that they receive. The biological neuron sums all the stimuli received via the dendrites (either from stimulating or inhibiting signals), and then this information is processed internally to decide whether to activate the output and stimulate the next neuron via the axon or not [23]. The biological neuron works on an all-or-nothing basis [24]. The neuron will trigger and fire the maximum output at once to the following neurons through the synapsis (the connections between neurons) if the stimuli that it is subjected to are higher than its specific threshold.

The artificial neuron sums all the inputs that it receives from the previous layer and provides it to the activation function, which computes the output and sends it to the next layer. Older versions of the artificial neuron were even more similar to the biological one because they used a step activation function that closely mirrored the all-or-nothing behavior of the biological neuron.

Additionally, both can improve themselves. The biological neuron can do it via synaptic plasticity: a process that allows the strengthening or weakening of the synapsis based on their activity [25]. The artificial neuron, instead, can do it via backpropagation, updating its internal parameters based on the feedback received regarding its performance during training.

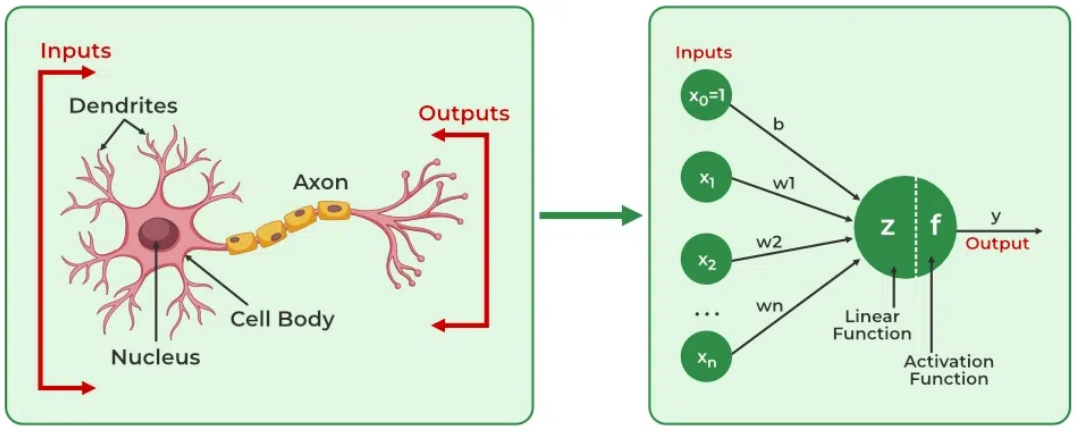


Figure 1.3: A representation of the natural and artificial neurons that captures the structural similarities between them.

The main strengths of a neural network are its adaptability to different tasks and its ability to improve by learning from its mistakes.

Instead of relying on hard-coded rules set by a programmer, a neural network can use its input data to compute the desired output through an iterative process. During this process the network computes the desired output and then receives feedback on the quality of its output, changing its internal processes on its own.

A neural network is a non-linear function made up of simple elements, called neurons (or nodes). The input and the output can be considered vectors of dimension equal to the number of nodes in their layer. Each neuron is connected to all the other neurons from the previous layers to acquire input data and feeds it to the neurons of the next layers. Internally, each input is multiplied by a weight and summed to a bias, and the output is fed to the neurons in the successive layers. This is the structure of a feed-forward neural network. If a NN achieves enough depth through the stack of a sufficient number of layers[4], it can be referred to as a Deep Neural Network (DNN). Deep Learning is the discipline of the study and development of DNNs.

Training is the iterative process that lets a NN learn from its mistakes and tunes its behavior to reflect the desired response. There are two main techniques to train a neural network: supervised and unsupervised training. Supervised training involves the use of labeled data, this means that the desired prediction is provided to the neural network. This kind of approach is interested in finding the relationship between the input and output data, to provide an accurate prediction. Supervised learning is most commonly used for classification and regression tasks [27].

Unsupervised training uses unlabeled data, this way the network does not know what the prediction should be. It is mainly used to discover new patterns and relationships in the data. It can be used for exploratory data analysis and clustering [27].

Other techniques are also available, such as reinforcement learning, that rewards or punishes a network respectively for desirable or undesirable behavior [28][23], and semi-supervised learning that blends supervised and unsupervised learning.

[4]There is no universal consensus on the amount of hidden layers required to separate a shallow neural network from a deep one, although most researchers in the field consider a neural network as deep if it has >2 non-linear layers and very deep if it has >10 non-linear layers [26].

1.4.1 The neuron

The neuron (or node) is the unit of the feed-forward neural network [29]. It takes multiple inputs x_i, computes the output y_k and then shares it with the following layer. Its behavior is characterized using the following function:

$$y_k = f\left(b_k + \sum_{i=1}^{n} x_i w_{ik}\right)$$

- Input x_i: input data for the neuron. Acquired from the previous layer, or from the input layer itself.
- Output y_k: the result of the activation function. It will be used as an input for the following layer or as the definitive prediction if is the last layer.
- Weights w_{ki}: tuneable parameters that change the behavior of the neuron. The tuning of the weights is the main way in which the training process can improve the neural network. Each neuron has multiple unique weights, one for each input.
- Bias b_k: a tuneable offset for the input of the activation function.
- Activation function f: characterizes the response of the neuron itself, dramatically changing its behavior. It is usually non-linear. The most common activation functions are the sigmoid, the hyperbolic tangent, and the rectified linear unit. A detailed explanation will be provided in subsection 1.4.4.

A depiction of the structure of the neuron can be seen in Figure 1.4. Even if the neuron itself has a simple structure, the amount of them can cause a DNN to balloon in computational complexity. For example, the ResNet-152 contains more than 19 million parameters [30].

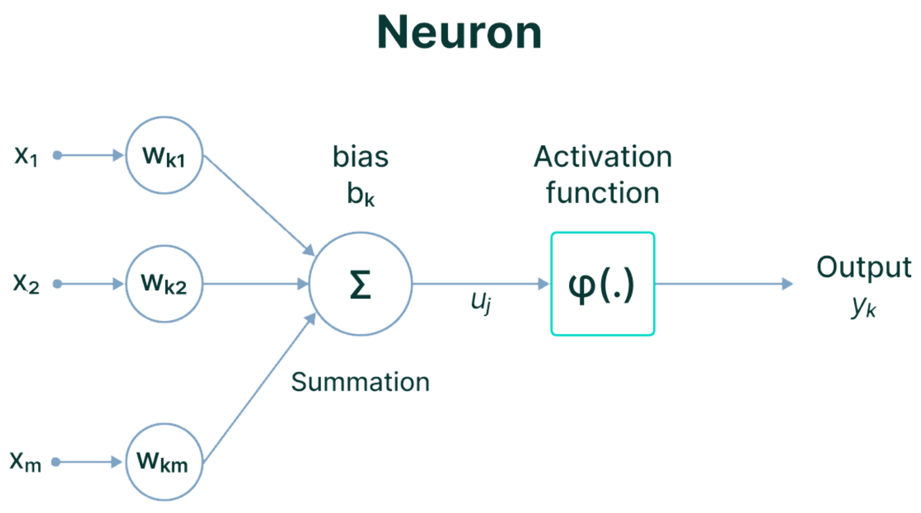


Figure 1.4: The structure of a neuron.

1.4.2 Architecture of the feed-forward neural network

The feed-forward neural network is the earliest implementation of deep learning. This kind of neural network is called feed-forward because the information flows forward, from the input layer, through all the hidden layers and to the output layer [29] (Figure 1.5).

The amount of nodes in a layer is called the width of a layer, while the amount of layers is referred to as the depth of the network. The input layer is used to pass the input data to the network. The output layer contains the final prediction of the network. The hidden layers are the intermediary layers between the input and output layers. In these, there are the units responsible for processing the data and the application of non-linear functions.

The width of the individual hidden layers may vary depending on the structure of the network. Different structures provide different properties. For example, in the autoencoder (Figure 1.6) the output corresponds to the input and narrows the width of the layers at the center of the network. The shrinkage forces the network to learn a compressed representation of the original data through the minimization of the reconstruction error. This causes the network to perform a data dimensionality reduction, learning the meaningful features of the data and discarding the useless ones [31]. These properties can be used for multiple purposes, such as data denoising, data representation and clustering [32].

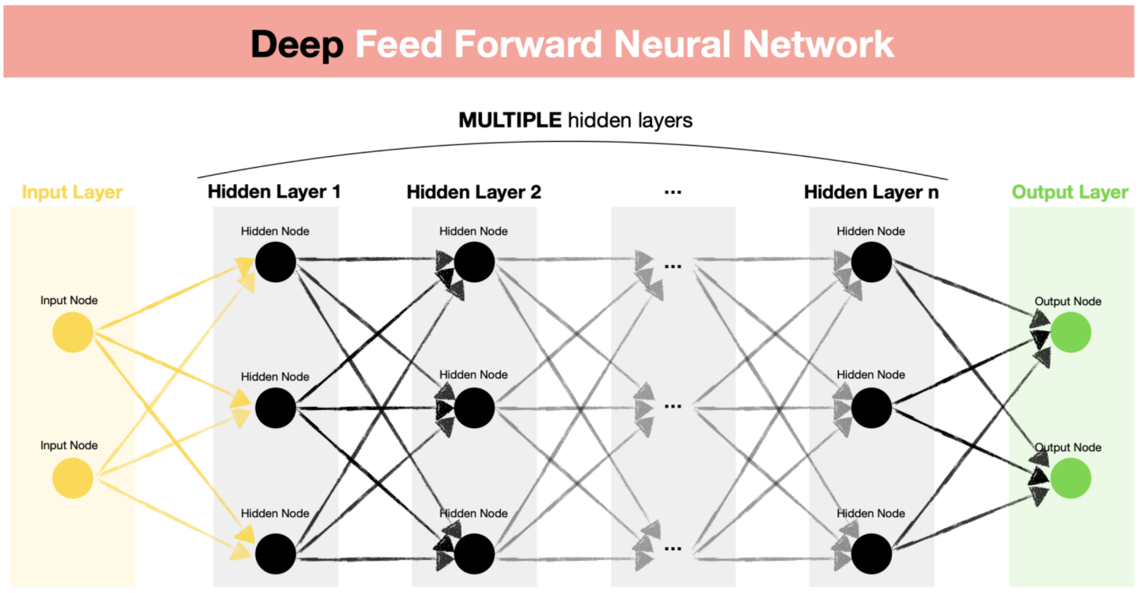


Figure 1.5: Feed-forward network structure. The input layer passes the input data to the network, while the output layer contains the final prediction. Most of the computations happen in the hidden layers.

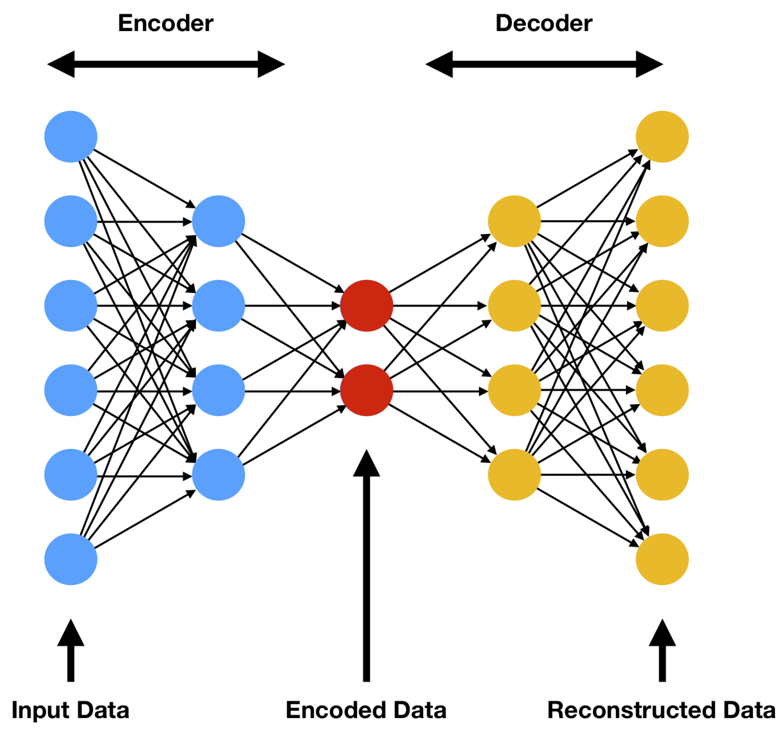


Figure 1.6: Autoencoder network structure. Both the encoder (shrinking width) and the decoder (widening width) components can be seen in cascade.

1.4.3 Training process

The training process, in supervised learning, is the iterative process that tunes the internal parameters of a neural network through the feedback received on its prediction. To provide the network with this feedback, its prediction and the ground truth (the data containing the correct expected prediction) must be confronted. To do so, a loss function is required. The objective of the training process is the minimization of the loss of the network, to align its predictions as much as possible to the ground truth.

In practice, the dataset is divided into three different subsets: training data, validation data, and testing data. The training data is the data used to train the network, the validation data is used to train hyperparameters (parameters that regulate the learning process and not the network itself) and to evaluate the capabilities of the network during training, and the test set is used to assess the network's performance. This means that the processes related to training will only be applied to the training set.

Loss functions

The loss function $\mathcal{L}$ is the function that computes the loss, a value that indicates how much the prediction was far from the ground truth (lower is better). The choice of the loss function is critical for the correct training of the neural network and is based on the structure of the dataset and the task. The tasks performed by neural networks can be broadly divided into two groups: regression tasks and classification tasks.

A regression task aims to predict a continuous value [33], through the approximation of a mapping function [34]. Some examples of regression tasks are image denoising [35] and deblurring[36]. One of the most used loss functions for regression tasks is the Mean Squared Error (MSE):

$$\mathcal{L}_{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

- n: number of samples.
- y_i: true value.
- $\hat{y}_i$: predicted value.

The MSE can penalize big errors heavily, if this behavior is not desirable another loss function should be considered. This makes the MSE more susceptible to outliers in the data and makes its behavior less interpretable because it uses the squared error [29].

In the classification task, the goal is to identify the correct class of a given sample from predefined N classes [34]. One of the most popular loss functions for binary classification (classification between two classes, 0 and 1) is the binary cross entropy (BCE):

$$\mathcal{L}_{BCE} = -\frac{1}{n}\sum_{i=1}^{n}\left[y_i \cdot \log(\hat{y}_i) + (1 - y_i) \cdot \log(1 - \hat{y}_i)\right] \tag{1.4}$$

- n: number of samples.
- y_i: true label, 0 or 1.
- $\hat{y}_i$: probabilistic prediction, between [0,1].

Usually, NNs used for classification do not output directly a class label, but a probabilistic value between [0,1]. The BCE works very well with this kind of output because it can regulate the loss according to the confidence of the network. For example, consider a training label with the target 1, and the output of 0.6. In this instance, the model has a 40% confidence in the wrong result. This loss function penalizes this 40% by returning the loss value $-[1 \cdot \log(0.6) + (1-1) \cdot \log(1-0.6)] = -\log(0.6) = 0.22$. In the case of a perfect prediction the loss function would output a loss of $-\log(1) = 0$ [37].

Backpropagation

We can consider the loss function as a n-dimensional function where n is the total amount of parameters in the neural network (weights and biases). The randomly initialized parameters represent the starting point (the black dot in Figure 1.7), and almost certainly, they will not yield the best results. To solve the learning problem, we must find the combination of weights and biases that minimizes the loss function [38].

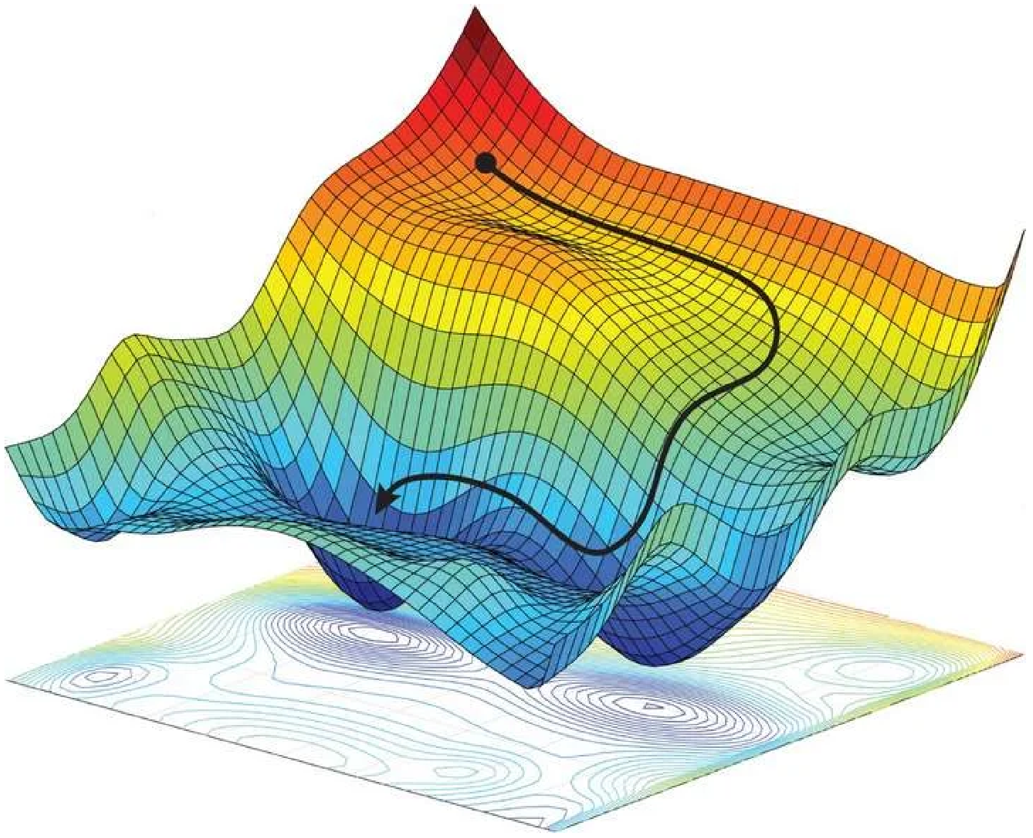


Figure 1.7: An intuitive visualization of gradient descent. From every starting point, following the opposite of the gradient leads to the minimum of the loss function.

To reach the minimum of this function we can go in the opposite direction of its gradient[5] $\nabla\mathcal{L}$:

$$\nabla\mathcal{L} = \left(\frac{\partial\mathcal{L}}{\partial w_1}, \frac{\partial\mathcal{L}}{\partial w_2}, \ldots, \frac{\partial\mathcal{L}}{\partial w_i}, \frac{\partial\mathcal{L}}{\partial b_1}, \frac{\partial\mathcal{L}}{\partial b_2}, \ldots, \frac{\partial\mathcal{L}}{\partial b_i}\right)$$

Usually, the calculation of the gradient would require the forward propagation algorithm. The conventional approach requires an iteration of the following computation for each weight w_i and bias b_i:

$$\frac{\partial\mathcal{L}}{\partial w_i} \approx \frac{\mathcal{L}(w + \epsilon e_i) - \mathcal{L}(w)}{\epsilon}$$

where $\epsilon > 0$ is a small positive number and e_i is the versor in the i^{th} direction. This way it is possible to estimate $\frac{\partial\mathcal{L}}{\partial w_i}$ by computing the loss $\mathcal{L}$ for two slightly different values of w_i. The same reasoning can be applied for the biases [39]. This approach is very slow due to the explosive computational costs tied to the requirement for an iteration for each single parameter.

[5]This means that the loss function must be continuous and differentiable. To use backpropagation the use of continuous and differentiable activation functions is mandatory. Their discontinuity is one of the main reasons because loss functions like the step function fell out of favor with the advent of deep learning [38].

A faster way to obtain the gradient would be to use the backpropagation algorithm. With this algorithm, the derivatives are obtained starting from the last layer of the model and progressing backward to the first with the use of the chain rule. In this manner, it is possible to compute all the derivatives in one fell swoop with minimal computational cost (can be considered equal to 1 or 2 forward propagation iterations [39]), facilitating the efficient training of deep neural networks.

After the gradient has been computed, each weight (and bias) is updated using the following equation:

$$\Delta w_i = -\gamma \frac{\partial \mathcal{L}}{\partial w_i} \quad \text{for} \quad \text{i=1,...,l},$$

where γ represents a learning rate, i.e. a proportionality parameter that defines the step length of each iteration in the negative gradient direction [38].

Unfortunately, the backpropagation algorithm tends to converge to the local minima, but with some tricks (like momentum) it is possible to induce the reach of the absolute minimum or lower local minimums. Due to the properties of the backpropagation algorithm, the training speed is proportional to the activation of the neuron. This means that the parameters of an unresponsive neuron cannot be changed and that a saturated function is a slow learner [39].

Training optimizers

To compute an accurate gradient of the loss function all the data should be considered, but this means that we can make only a step each epoch. This approach is very methodical but also very slow. The epoch is a metric used to measure the training progress. One epoch represents the iteration over all the training set during the training process, this means that each training sample had the opportunity to change the internal model parameters. To train a neural network, multiple epochs are required.

To speed up the training process we could take faster approximate steps rather than slower perfect steps. To do so we can use the Stochastic Gradient Descent algorithm (SGD). Instead of using all the training data to calculate the gradient, SGD considers only a batch of data. A batch is a random selection of n data points (n is fixed during training and is called batch size), that are never been used for training in this epoch. If the batch size is sufficient and its data are representative of the entire data set, the step will be a good approximation of the exact gradient [40].

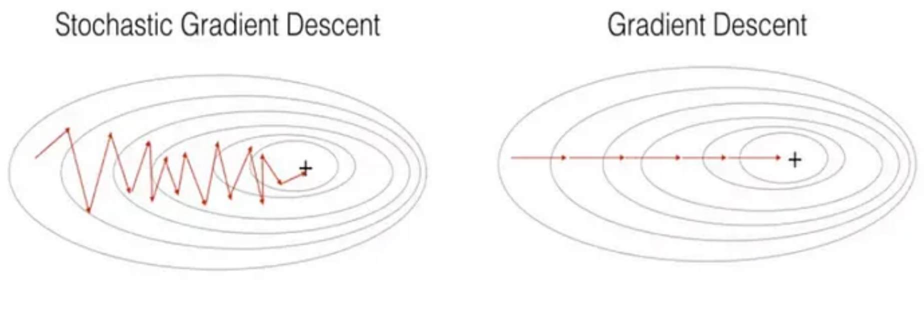


Figure 1.8: The overall amount of steps required to reach the minimum is higher, but due to the enhanced speed, SGD can obtain a comparable result in less time.

As seen in Figure 1.8, the biggest issue of SGD is the repetitive oscillation of the trajectory. These oscillations slow down the training imposing a higher amount of steps and force the use of lower learning rates to keep an acceptable stability. To mitigate these problems, we can implement in the SGD algorithm a technique called momentum. Like the physical counterpart, we can think of the gradient $\nabla\mathcal{L}$ as an acceleration that changes the step length, like a velocity. The momentum slows down and reduces the steps on bumpy surfaces and accelerates on plateaus [23] (as seen in Figure 1.9). Analytically, the momentum accumulates an exponentially decaying moving average of past gradients and continues to move in their direction [41]. This behavior can also help improve the final results, leading the parameters to a lower minimum than the one that would have been reached with conventional gradient descent. The equations for SGD with momentum damping are the following:

$$m_k = \beta m_{k-1} + (1-\beta)\nabla\tilde{\mathcal{L}}_k$$

$$w_{k+1} = w_k - \gamma m_k$$

where m is the momentum, $\nabla\tilde{\mathcal{L}}$ is the approximation of the gradient, β is the damping (or forgetting rate) that controls the number of steps in which the information is kept relevant, usually $\beta = 0.9$, w is a generic parameter of the network (weights and biases), γ is the learning rate and the superscript specifies the step [42]. The implementation of momentum makes the trajectory more stable and allows the use of higher learning rates, speeding up the training process.

Many other optimizers are available like RMSProp [43], AdaGrad [44], and Adam [45]. All of them have different properties and different levels of complexity. The most used optimizer is the Adam (a simplified name for adaptive moment estimation) because it provides the benefits of both RMSProp and AdaGrad. The Adam optimizer computes adaptive learning rates using the moving first moment (the average M_k) like RMSProp and computes the moving average of the second moment of the gradient (the uncentered variance V_k). Each step is computed as follows:

$$M_k = \beta_1 M_{k-1} + (1-\beta_1)\nabla\tilde{\mathcal{L}}_k$$

$$V_k = \beta_2 V_{k-1} + (1-\beta_2)\nabla\tilde{\mathcal{L}}_k^2$$

Where $\nabla\tilde{\mathcal{L}}_t$ is the gradient of the current batch and β_1 and β_2 are dampening hyperparameters, usually they are set to $\beta_1 = 0.9$ and $\beta_2 = 0.999$. Therefore, each update to the weights is provided through the following equation:

$$W_{t+1} = W_t - \frac{\gamma}{\sqrt{V_k + \epsilon}} M_k$$

Where ϵ is a small value, usually $\epsilon = 10^{-8}$, used to prevent a possible division by 0 [29], [45].

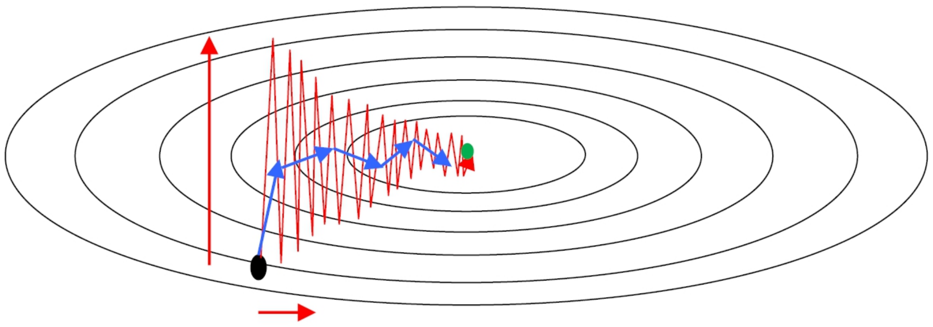


Figure 1.9: A visual representation of SGD with momentum. The red line is the SGD trajectory, while the blue line is the SGD with momentum trajectory. The effect of the momentum is visible, dampening the movement where the gradient is higher and accelerating it where the gradient is lower.

Training, validation and test sets

To train a neural network, not all the data is considered in the same way. To achieve a good performance, promote generalization and a fair evaluation, the total dataset is split into three parts: the training set, the validation set and the test set, their ratio depends on the size of the dataset. These three sets have different objectives:

- Training set (70-80% of the dataset): is used to train the network.
- Validation set (10-15% of the dataset): is used to tune the hyperparameters [46] and as an intermediate evaluator, to keep track of the mean epoch loss.
- Test set (10-15% of the dataset): is used to test the performance of the model. The test set must remain unseen from the model to provide an unbiased evaluation of its performance.

There are some techniques that mix the test and validation set to provide more diverse training data to the network, this process is called cross-validation. The most used cross-validation technique is the k-fold cross-validation. In k-fold cross-validation k different subsets of the validation and training set take turns to be the validation set. In each epoch a different subset has the role of cross-validator [47]. K-fold cross-validation is often used to optimize the use of the data for training purposes because the network can be trained on a bigger set of data.

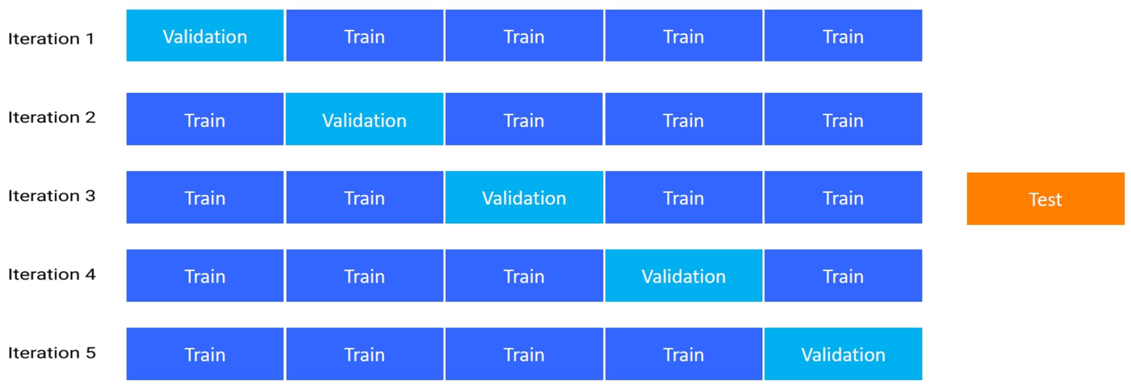


Figure 1.10: The routine of a k-fold cross-validation with $k = 5$.

1.4.4 Activation functions

The right activation function has a deep impact on the performance of a neural network [23], [48]. The standard choice is the use of a non-linear function, because it adds non-linearity to the network, this way the prediction varies non-linearly based on its explanatory variables, allowing the solution of non-linear problems [29]. Furthermore, the use of a linear activation function would neutralize the use of multiple layers, making them behave like a single layer, because linear functions added together just make another linear function.

The following activation functions are the main ones used for the training of neural networks; their behavior is shown in Figure 1.11.

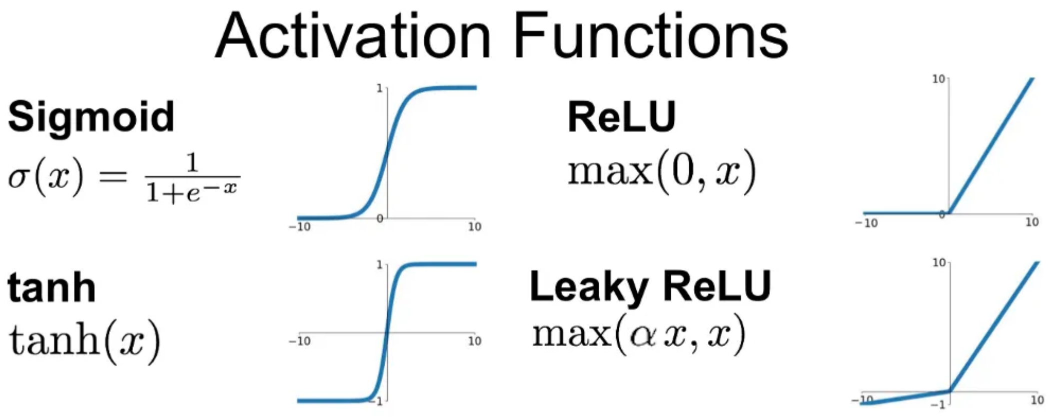


Figure 1.11: The most common non-linear activation functions.

Sigmoid

The sigmoid has been used from the early days of neural networks [49], [50], is characterized by the following equation:

$$f(x) = \frac{1}{1+e^{-x}}$$

Since the output is bounded between [0,1] it is usually used for binary classification purposes, because this configuration allows the network to express its confidence in the prediction. A variant of the sigmoid, called softmax, can be used to estimate the probability of data belonging to a particular class in multi-class classification, because the output probability for each category always sums up to 1 [34]. The equation for the softmax is the following:

$$f(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{N} e^{x_j}} \quad \text{with } i \in [1, \ ..., N]$$

The behavior of the sigmoid can cause some problems: due to a $f'(x) < 1$ far from its center, the function can cause a vanishing gradient, a gradient so close to 0 that can inhibit any further training during backpropagation [51], saturating the output for higher and lower inputs. It is also not zero-centered, slowing the learning process [49]; this problem can be reduced if the input data itself is zero-centered. Another problem is its computational complexity, which can slow down the training process [50]. Due to these issues, currently, the sigmoid is not used often in the hidden layers of a deep neural network, but it can be safely employed as a last-layer activation function since these problems require multiple layers to manifest.

Hyperbolic tangent (*tanh*)

The hyperbolic tangent (*tanh*) is similar to the sigmoid, and its behavior is characterized by the following equation:

$$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$$

The *tanh* overall behaves better than the sigmoid: it is zero-centered, with an output bound between [-1,1] and has a bigger derivative in its domain, speeding up the training process.

Even if, for certain aspects, the *tanh* can be considered better, it still has some problems. The *tanh* still suffers from the vanishing gradient problem due to its bound nature, and its computational complexity can slow down the training speed [50].

ReLU

The ReLU (Rectified Linear Unit) is the most used activation function in deep learning. Its behavior is characterized by the following equation:

$$f(x) = \max(0, x)$$

The ReLU is widely used due to its flexibility, simplicity, and superior training performance owing to its constant derivatives identity for positive and negative values.

$$\begin{cases} f'(x) = 1 & \text{if} \quad x \geq 0 \\ f'(x) = 0 & \text{if} \quad x < 0 \end{cases}$$

The $f'(x) = 0$ for the negative values means that weights and biases are not updated, while the $f'(x) = 1$ provides consistent and predictable gradients. The ReLU is mainly used to avoid the vanishing gradient problem, it alleviates it since is not bounded in one direction [51], and its simplicity accelerates the training process [52]. Additionally, its zero values provide representational sparsity to the network, because only a small number of neurons is active at a time. This encourages the learning of more informative features in the data and a better invariance to input changes.

This activation function has still some drawbacks. It is susceptible to the dying ReLU problem, a special case of the vanishing gradient problem. This can happen when a big bias is learned causing the neuron to never activate, not contributing anymore to the network's output, it can often happen if the learning rate is high [53]. The problem can be contained with an appropriate random asymmetric weight initialization or the use of additional parameters.

Leaky ReLU

The leaky ReLU is a variant of the ReLU that tackles the problem of the dying ReLU via the addition of a small slope, called α, for negative values. α can be chosen a priori (usually $\alpha = 0{,}01$) or implemented as a learned parameter that is adjusted during the learning process (in that case the activation function is called Parametric ReLU). Its behavior is characterized by the following equation:

$$f(x) = \max(\alpha \cdot x, x)$$

This behavior provides the network with better expressing capabilities since there are no "dead" nodes in the network [54], but its effects may be unstable [53]. The leaky ReLU is also more complex and expensive to run than the ReLU, so it can negatively impact the training speed.

1.5 Techniques for performance improvement

After covering the basic requirements to make a neural network work, we will investigate the use of more advanced techniques that may help to compensate for the weaknesses and enhance the strengths of neural networks. The following methods will be used to improve the overall performance of the neural networks created for this project.

1.5.1 Residual connections

To train effectively a DNN it is important to take precautions against the vanishing/exploding gradient problem. One of the most effective ways to do so is using residual connections. Residual connections allow connections that skip layers adding the output of a layer deeper into the network [55].

This connection also improves the generalization capabilities of the network and prevents chaotic behavior [55]. Residual connections are powerful and computationally inexpensive tools but require the two connected layers to be of the same size. This is usually not a problem for CNNs because they can very easily control the shape of the data, but it is more challenging for feed-forward NNs. If an architecture uses frequent width changes, the implementation of residual connections is not straightforward, and this requirement must be considered during the design of the structure of the network.

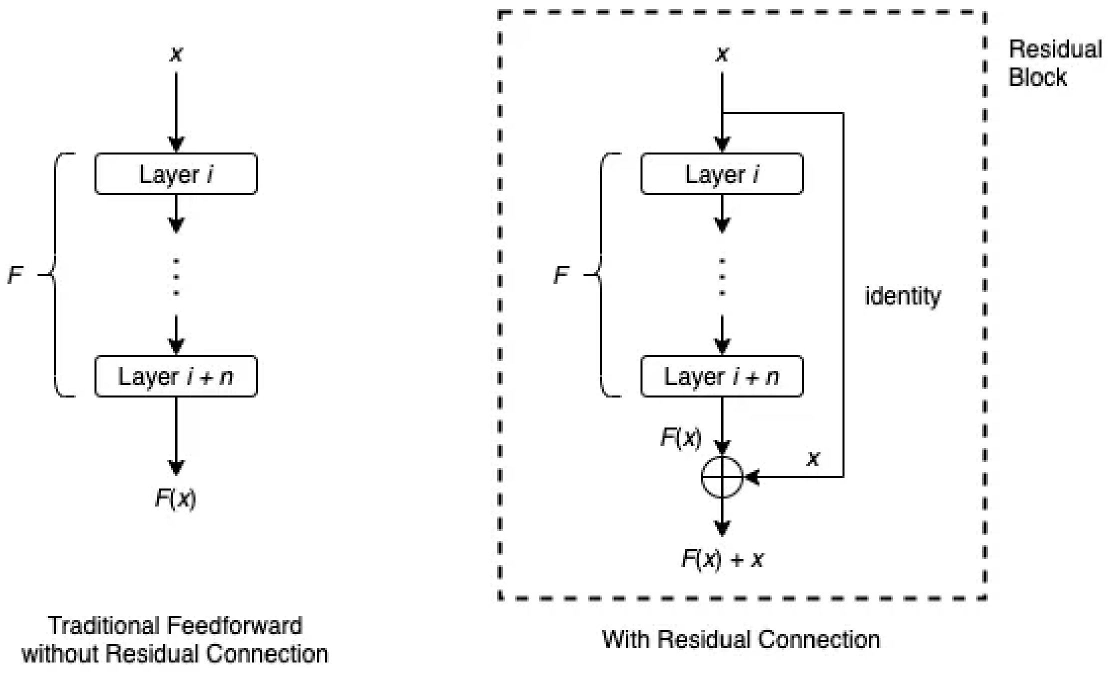


Figure 1.12: The representation of a traditional feed-forward neural network and a feed-forward neural network with a residual connection. The final output is the output of the last layer added to the input of this portion of the neural network.

1.5.2 Batch normalization

Batch normalization is a process that normalizes the inputs to a layer for each mini-batch of data [29]. The normalization of the output data of a layer improves the computation of the following layer, preventing the change of distribution of each layer's input and stabilizing the training process. Furthermore, batch normalization allows for further leeway during the initialization of the weights and reduces the need for dropout [56].

1.5.3 Dropout

The dropout controls the activation of each single neuron, when implemented it can deactivate a portion of the neurons, chosen randomly, in the next layer. The amount of neurons affected can be selected and tuned, and occasionally, different layers may require different dropout values due to different overfitting risks. A neuron that dropped out does not make computations and does not learn. It is usually used to prevent overfitting and to create a more resilient neural network that distributes learning equally between its neurons.

Dropout can be considered a computationally cheap alternative to a model ensemble because all the neurons are effectively trained on slightly different data [29].

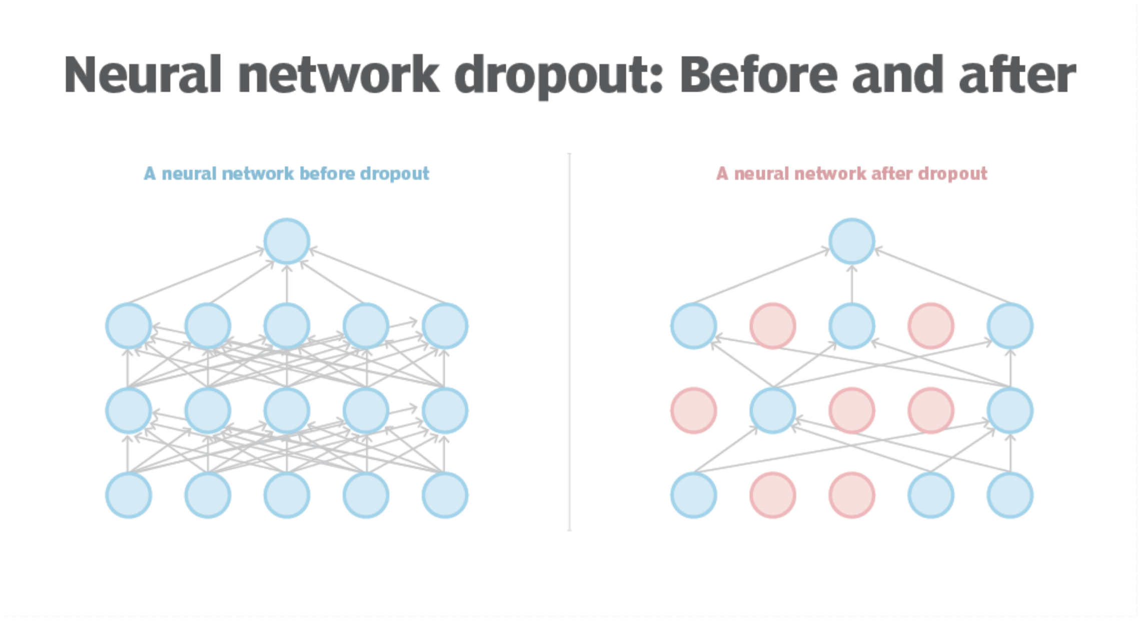

Neural network dropout: Before and after
A neural network before dropout
A neural network after dropout

Figure 1.13: The dropout shuts off the connection and the computation of a random selection of a predefined amount of neurons. This compels the network to train redundancy in its system, to always provide satisfactory results.

1.5.4 Dynamic learning rate

The learning rate is one of the most important hyper-parameters for the training of NNs, choosing the right one is very important to obtain good results. A learning rate that is too high may cause instabilities in the learning process, while a low learning rate may slow down training significantly. To provide a good middle ground and achieve initially the speed of a big learning rate and the precision of a small learning rate later during training, we use a dynamic learning rate. A dynamic learning rate is a learning rate that changes with the amount of steps that are done during training. The most common implementation uses an exponential decay learning rate scheduler that at a set interval of steps computes the new learning rate γ_{i+1} by multiplying the previous one γ_i it by a coefficient α:

$$\gamma_{i+2} = \alpha \cdot \gamma_i \quad \text{with} \quad 0 < \alpha < 1$$

This ensures an exponentially decaying learning rate that can adapt better to the shape of the loss function, improving training [23].

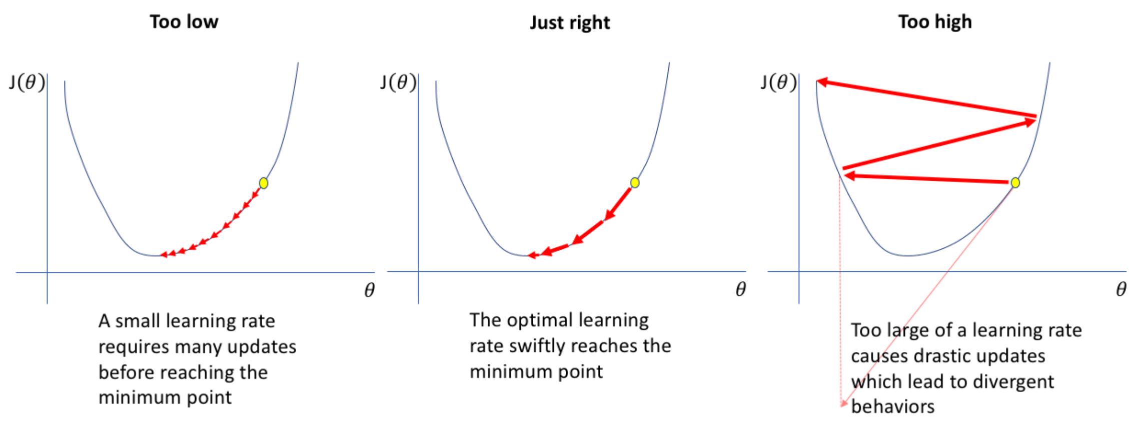

Too low
Just right
Too high
J(θ)
θ
A small learning rate requires many updates before reaching the minimum point
The optimal learning rate swiftly reaches the minimum point
Too large of a learning rate causes drastic updates which lead to divergent behaviors

Figure 1.14: The dynamic learning rate is bigger at the start, to speed up training and slows down after a while to be more precise around the features of the loss function.

1.5.5 L1 and L2 regularization

Regularization is a process that greatly reduces variance and promotes the development of a simpler model at the cost of a small amount of additional bias [57]. Regularization improves neural networks because they often suffer from a high variance that leads to overfitting and the impact of the small amount of additional bias is negligible compared to the improvements. The L1 and L2 regularization are two regularization techniques used to penalize detrimental parametrical changes to keep a stable and efficient training process. L1 regularization, or Ridge regression, adds the following penalization:

$$\mathcal{L}_{reg} = \mathcal{L} + \lambda_1 \sum_{i=1}^{n} |w_i|$$

- $\mathcal{L}_{reg}$ is the total loss after regularization
- $\mathcal{L}$ is the loss
- w_i are the parameters of the neural network (weights and biases)
- λ_1 is the coefficient used to control the impact of the Ridge regression on the total loss.

L1 regularization applies a penalization to the modulus of the sum of all the parameters. This penalty pushes down all the weights and biases and reduces the overall complexity of the network.

L2 regularization, or LASSO regression (Least Absolute Shrinkage and Selection Operator regression), adds the following penalization:

$$\mathcal{L}_{reg} = \mathcal{L} + \lambda_2 \sum_{i=1}^{n} (w_i)^2$$

- $\mathcal{L}_{reg}$ is the total loss after regularization
- $\mathcal{L}$ is the loss
- w_i are the parameters of the neural network (weights and biases)
- λ_2 is the coefficient used to control the impact of the LASSO regression on the total loss.

L2 regularization applies a penalization to the sum of all the squared parameters. This penalty prevents the accumulation of value on single parameters promoting an equal distribution. The equal distribution improves the stability and the generalization capabilities of the network.

Together L1 and L2 regularization can be used effectively to prevent overfitting. Since they are linear functions they can be computed together at once:

$$\mathcal{L}_{reg} = \mathcal{L} + \lambda_1 \sum_{i=1}^{n} |w_i| + \lambda_2 \sum_{i=1}^{n} (w_i)^2$$

1.5.6 Oversampling and undersampling

Another method to counter an unbalanced dataset is to use oversampling or undersampling to improve the quality of the prediction [58]. Oversampling randomly populates the minority class with samples (copied or synthesized) to rebalance the dataset, while undersampling focuses on excluding data from the majority class from the dataset. The simplest forms of resampling are random. During random oversampling, a random selection of minority data is duplicated via bootstrapping (random selection with replacement), conversely during random undersampling a random selection of majority data is excluded via bootstrapping.

A more sophisticated approach to oversampling involves the creation of synthetic data points using the ones that are already present in the dataset. SMOTE (Synthetic Minority Oversampling Technique) [59] is a technique that populates the dataset with synthetic data similar to the minority class, by operating in the feature space of the data. To oversample, the algorithm considers a sample (feature vector), its nearest neighbor in the feature space and then computes the difference. The difference is then multiplied by a random number between 0 and 1, the final value will be added to the original sample to create a new synthetic sample [59]. This process improves generalization due to a more representative minority class.

To promote an unbiased evaluation of the network's performance, the validation and test datasets will keep the original class distribution.

1.6 Objectives

At this point in the development of Argot, the research team is trying to improve its performance over what is normally possible with conventional methods based on similarity, sequence, or structure. One element of Argot that can be improved is its prediction ranking system. When Argot outputs a prediction it also outputs other useful information that may be used to assess the reliability of the prediction. Currently, the method used to rank the predictions involves only the total score ignoring all the other potentially useful data. The first objective is to create a system that can leverage this additional information to provide a better performance than the previous system.

Another interesting element of the Argot predictions is the characteristic of occasionally predicting apparently wrong functions that with time are corroborated by the literature, anticipating the experimental evidence. This property is worthy of investigation, especially if these predictions can be discerned from the other wrong predictions. The second objective consists in the implementation of a system that can reliably recognize them.

This project has two objectives. To reach these goals, two different neural networks have been implemented.

Main Net

The first objective is to create a system for the evaluation of the predictions of Argot 2.5. To do so, we created a neural network capable of taking the Argot 2.5 output as input and predict the true simgic of the prediction with respect to a ground truth. The simgic is directly correlated to the quality of the prediction, so it can be used as a metric to evaluate the prediction itself [12]. From now on, this neural network will be referred to as Main Net.

Future Net

The second objective stems from the desire to harness the predictive capabilities of Argot to suggest the future direction of experimental investigation of gene products; Argot and other predictive tools may have the capability of anticipating experimental evidence regarding GO annotations.

The field of molecular biology is plagued by the problem of missing information. There are lots of protein functions that are still unknown, so it is not correct to consider every inconsistency in results as wrong, because it may as well refer to some yet undiscovered function. This is very important in the field of GO predictions because it means that some predictions that are considered unsubstantiated by the current evidence may be proved right in the future. It has been observed that some false positives became true positives once the updated database provided a more complete annotation of the inquired gene products [60].

A neural network is tasked to find what separates a mere false positive from a predictive breakthrough in the annotations. This neural network may be used to direct experiments for the confirmation of the most plausible annotations. From now on, this network will be referred to as Future Net.

1.6.1 Strengths and weaknesses of deep neural networks

The decision to use neural networks as the tool for this project has been informed by multiple factors. DNNs have multiple benefits to their use, these are the main ones and how we can exploit them for satisfactory results:

- DNNs are very flexible and can provide good performances for most tasks if tuned properly since they can compute any function [39]. Their flexibility allowed us to achieve good results in both the predetermined objectives, with minimal changes to the networks.
- DNNs are resilient to perturbations, artifacts and variations in the input [61]. This aspect is very useful because most of the dataset is made of low-quality data. Most of the available ground truth is derived electronically (over 99%), with only a small minority of the data corroborated by experimental evidence. Thanks to DNNs capabilities, a high amount of noisy data is still very valuable [62], [63] for training purposes, so is still possible to reach good performances (this process is called robust learning).
- DNNs can evaluate the confidence of their own prediction, this means that we can use thresholding[6] to further improve the results.

DNNs also present some weaknesses. These are the most notable, and what has been done to tackle them:

- A lot of data is required to train the DNN to satisfactory levels of performance, especially if the model is complex, with a high amount of parameters. We used a dataset with more than 17 million original different data points, so we have access to sufficient training data.
- Neural networks are often considered black boxes. DNN explainability is a promising field that is being actively researched on [64], but at this time it is not rare for the creators of a DNN to not fully know all the intricacies of their model. This is a downside of this approach, but only a small part of the algorithm is a DNN, so most of what the algorithm does is well understood. Overall, the improvement in performance is still worth the trade-off.
- DNNs tend to misclassify data belonging to minority classes because their impact on the loss is often inferior to the majority class. We mitigated these risks with proven techniques like cost-sensitive learning and oversampling [65].
- DNNs are prone to overfitting. Regularization methods can be implemented to limit the impact of overfitting on the network like dropout and batch normalization layers [66].

[6]Thresholding is the process of finding the best value to use as a threshold to discriminate between two different classes. This is an iterative process in which multiple possible thresholds are put to the test to find the best one.

Chapter 2

Materials and methods

In this chapter, we will describe in detail how we plan to develop the networks required to satisfy the objectives of this project.

2.1 Dataset structure

The original dataset was made to train, test and validate Argot 2.5. The primary objectives were to:

- Ensure a comprehensive representation of GO terms.
- Include a sufficient number of proteins (and corresponding protein-GO associations) to make the dataset suitable for training.
- Represent protein diversity adequately.

Since our networks are working directly with the output of Argot, our dataset inherited the same properties, with a diverse representation of GO terms and an adequate quantity of data. The complete dataset is composed of 17408776 unique entries. Each entry is an output of Argot 2.5, containing the SeqID, GOID, ontology, information content, internal confidence, total score and group score of the prediction. A smaller dataset of the first 1000000 entries (called dataset 1M) has been made to iterate faster during prototyping and allow for a quicker tuning process. The choice of using the first 1000000 entries instead of a random sample is intentional, since Argot 2.5 produces blocks of prediction for a single gene product, choosing a cohesive sample allows for a reduced amount of gene products with all of the associated predictions instead of a random assortment of a lot of different gene products with very limited associated predictions. Keeping the same amount of predictions for each gene product between datasets is crucial for a consistent tuning of the network since a drastic change could make it unreliable. This method has been successful in keeping a similar ratio of predictions/SeqIDs between the two datasets because dataset 1M has an average of 60.20 annotations for each protein, while the full dataset has an average of 60.43.

Data quality

To train Argot, the GOA (release 2023-03-03) was fully restructured to be GO-centric instead of protein-centric.

For each GO term in the GOA, a list of associated proteins was created. Analyzing the evidence codes related to each annotation we learn that the data quality in the ground truth is not uniform, because it comes from three different sources: fully electronic or IEA (Inferred from Electronic Annotation), electronic assisted with UniProtKB and experimental.

0. Experimental annotations, non-IEA (48.2%): the experimental annotations are derived from experimental evidence and are the most reliable of all the annotations.
1. IEA derived from UniProtKB keywords (22.4%): electronic-assisted annotations that are created from the automated parsing of a published scientific paper containing the keyword, are considered significantly more reliable than other electronic annotations.
2. All remaining IEA annotations (29.4%): fully electronic annotations derived in silico, without the support of other systems. They can be considered the most unreliable type of data.

The proteins were sorted for reliability, resulting in a list of 29835 GO terms, each associated with a variable number of proteins (ranging from just one to several million proteins, depending on the frequency in which it appeared). The protein selection was performed through a clustering step based on sequence similarity, extracting up to 100 proteins per GO term (if a term had less than 100 proteins in the GOA it would be represented less than 100 times). After using CD-HIT [14] to ensure a shared sequence of no more than 60% the final dataset was created with 289817 unique proteins.

Notably, the proportion of experimental to electronic annotations in the final dataset differs significantly from the whole GOA dataset (IEAs are lower in our dataset compared to over 99.9% in the GOA). This intentional bias targets the uneven source of the data and aims to balance the origin of the annotations to avoid bias during the training phase of Argot.

Dataset split

In both of these datasets, the training/validation/test split is 70%/15%/15%. For consistency's sake, when k-fold validation is used, a k=6 is enforced, because it is the value that can induce the most similar split.

$$k\ validation\ split = (1 - test\ split) \cdot \frac{1}{k} = (1 - 0.15) \cdot \frac{1}{6} \approx 14.2\%$$

Ontology distribution

The three ontological categories of the dataset are unevenly split. The majority of the dataset is made of molecular functions, around $\approx 47\%$ of the dataset, while the cellular components and the biological processes trail behind with $\approx 33\%$ and $\approx 20\%$ of the dataset, respectively[1]. This uneven distribution may impact performance and, when relevant, the three ontologies will be analyzed separately. To keep intact intra-ontology properties of the data, during the tuning and training of the Main Net, each of the three sub-sets (one per ontology) has been scaled via standard scaling separately from the others. Hence, the data for each sub-ontology will have an average of $\mu = 0$ and a standard deviation of $\sigma = 1$, this approach contributed to improving the global performance. For the training and tuning of the Future Net, we will not make distinctions between ontologies, and we will apply standard scaling on all the data at once. These different approaches, tailored to each network, have been heuristically proven to improve results.

The ontological classification was implemented via one-hot encoding, which means that instead of using the data point "ontology" to specify the ontology of each entry, three different data points refer to the three ontologies. In this case, the data point identifying the ontology becomes 1 while the others remain 0. The categorization via one-hot encoding is good practice in the field of deep learning because it teaches the neural network that the three ontologies are mutually exclusive pieces of information, without the risk of extrapolating wrong information, something that would happen if it is used only one data point with a number between 1 and 3. Such data may imply to the network that one ontology is worth twice the other or that ontology 1 is the difference between ontology 2 and 3. This is because the neural network doesn't grasp the meaning of the data it only uses it for its own computations.

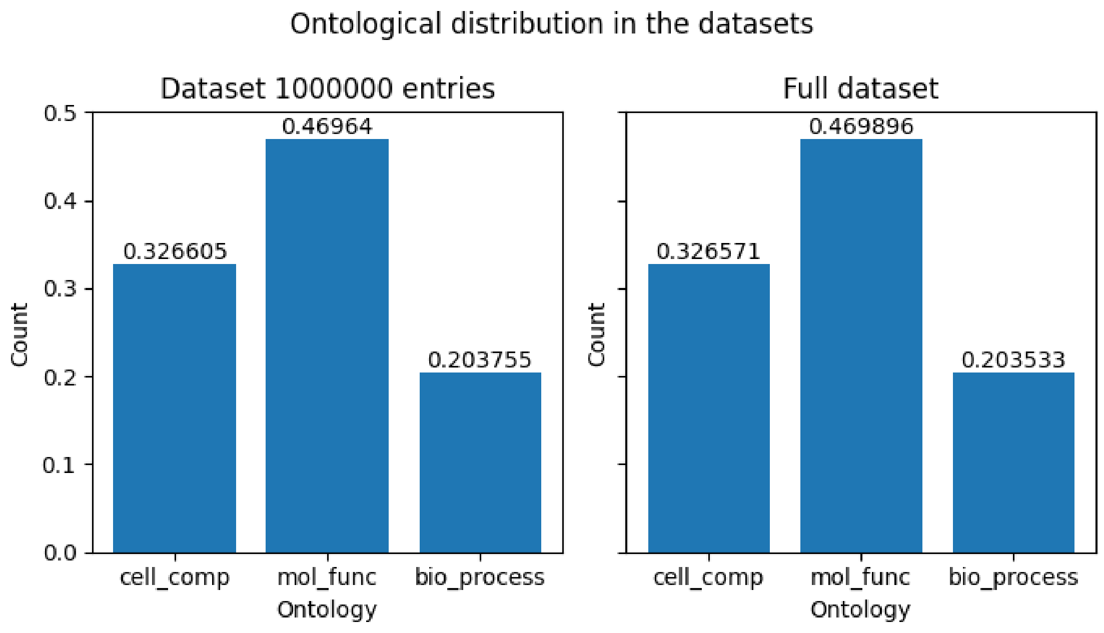


Figure 2.1: Distribution of the ontologies in both datasets. The ontologies are represented in the two datasets almost identically.

[1]Interestingly, this distribution is quite different from the distribution of the complete GOA. The values of the complete GOA are more evenly distributed with $\approx 32\%$ for the molecular functions, $\approx 31\%$ for the cellular components and $\approx 37\%$ for the biological processes [67].

Main Net-specific notes

The simgic is computed by comparing each prediction from Argot with the prediction in the ground truth; in this way, we can use it as a target metric to assess the quality of the prediction to train the Main Net. The simgic has a very skewed distribution with most of the data close to 0, and a small but relevant minority close to 1. Those are the most relevant predictions because they represent the times when Argot made a very good prediction. From now on, a prediction will be considered good if it yields a simgic value > 0.75.

The skewness of the simgic distribution is one of the most challenging aspects of training the Main Net, with a great impact on performance. Most of the techniques used have been implemented to contrast this aspect of the data.

During training, all the data points are treated equally regardless of the source. However, during testing, a specific test will be run to assess whether the Main Net behaves better with experimental data due to its higher reliability and quality.

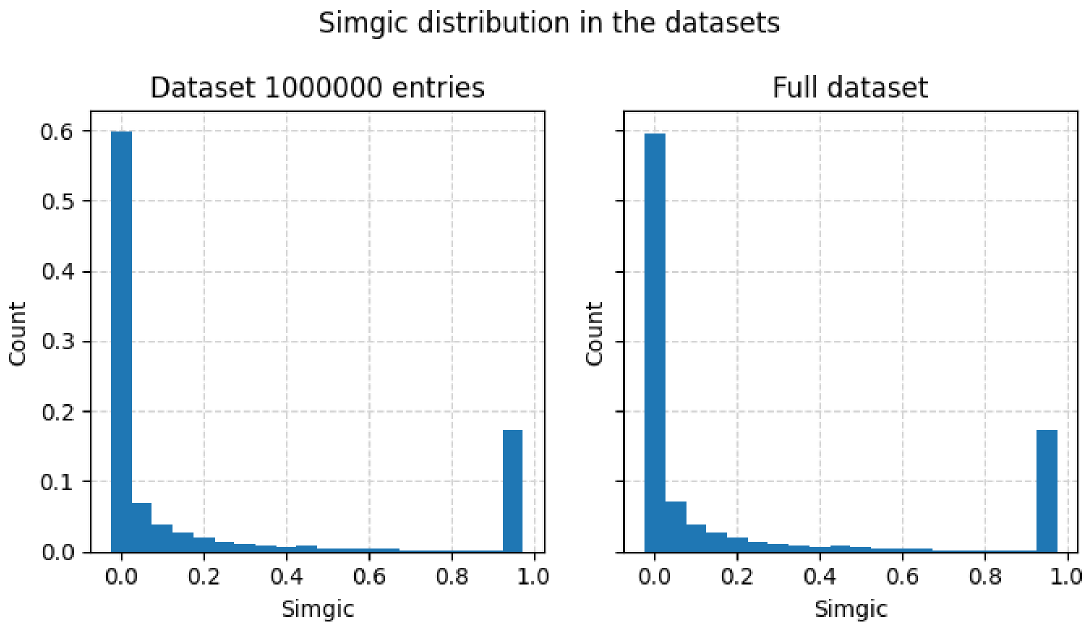


Figure 2.2: Distribution of the simgic in both datasets. The simgic are represented in the two datasets almost identically.

Future Net-specific notes

To train the Future Net the output data of Argot has been labeled as anticipatory 1 or not anticipatory 0. To provide this label we compared the ground truth from 2019 (gt2019) and an updated ground truth from 2023 (gt2023). To be labeled as 1, a prediction must bear an annotation added in the gt2023 and not present in the gt2019, regarding a preexisting gene product (as seen in Figure 2.3).

Even if label 1 is well-defined, label 0 is more flexible in its meaning. Due to the problem of missing information, we can't exclude that any of the entries labeled as 0 could be truly labeled 1, because the updated ground truth is not exhaustive. Label 0 means that there is no current evidence supporting its anticipatory nature.

The Future Net will be trained on a flagged version of the gt2023, the same used to train the Main Net.

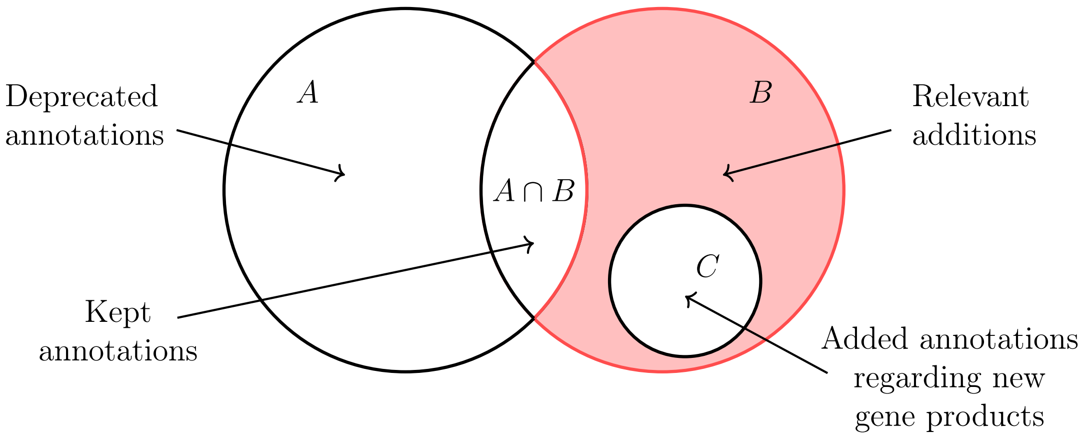


Figure 2.3: A Venn diagram representing the data inside the two versions of the ground truth. A is the gt2019, and B is the gt2023. $A \setminus B$ are the deprecated annotations, $A \cap B$ are the annotations that are shared between the two versions and C are the annotations that involve new gene products. All of these received the label 0. $B \setminus \{A \cup C\}$ is the set of new annotations regarding previously known gene products, these are labeled as 1.

2.2 Network and neurons structure architecture

The Main Net and the Future Net are similar in their architecture. Both are feed-forward deep neural networks with residual connections.

The choice of using a convolutional neural network (CNN) was considered as it is very well documented for a lot of uses, is easily scalable and has a proven track record of good performance [29]. Unfortunately, a CNN was not compatible with our use case because it requires single entries with vast amounts of data (it works best with image and time series classification), while the input we used had only 6 or 7 components, made of 3 or 4 data points (depending on whether the total score was considered) and the one-hot encoded ontology.

Since feed-forward NNs are less popular, there is also considerably less documentation for it, and almost none for use cases similar to our own. This lack of documentation means that no optimal procedures could be followed and that only general good practice and heuristics could be used to make most of the decisions.

Following, some of the decisions that concern both of the networks are described. The choices that regard only a network will be discussed in the appropriate subsection.

Technical tools

The development of the network and the associated scripts has been done in Python v3.9.2 [68]. The neural networks were developed using the Keras API v2.11.0 [69] and TensorFlow library v2.11.0 [70]. The plots were created with Matplotlib library v3.8.2 [71]. Data engineering and data analysis were done using pandas library v2.1.4 [72], NumPy library v1.25.2 [73] and scikit-learn library v1.3.2 [74]. All the Random Number Generation (RNG) seeds were defined, to provide maximum consistency during training and tuning and to improve reproducibility. Unfortunately, since most of the training was done using a GPU, its intrinsic RNG could not be addressed, which means that some of the training processes were still aleatory.

All the relevant code can be found in the following GitHub repository, `https://github.com/ggradara/neural_networks_for_Argot_25`.

Weight initialization

Weight initialization is a crucial topic for fast convergence and network stability. The weights of the network are initialized with random small values [0.0-0.1] to provide a good starting point for the gradient descent algorithm. The starting weights can't be all set to 0 because they would kill the gradient, blocking the training process.

The weight initialization method can vary a lot in their random distribution, changing their properties. A very common weight initialization method is the Xavier method, its often used to train neurons with *tanh* and sigmoid activation functions.

The Xavier method assumes the linearity of the activation function, a premise that is invalid for the ReLU and the Leaky ReLU, stalling the learning process in the neurons that use this activation function [75]. To train effectively nodes with ReLU and Leaky ReLU functions the He initialization can be used. This approach leads to better results [76] because the He initialization does not require this assumption, addressing the nonlinearity of the rectifiers.

Our networks use Xavier initialization for the nodes that use the sigmoid function and the He initialization for nodes that use ReLU and Leaky ReLU functions.

The layer structure of a single block

Even if we tested multiple different possible network architectures, with varying numbers of layers and widths we mostly used the same block to create the networks.

This block comprises of a dense layer of variable width, a batch normalization layer and a dropout layer, in this order. The dense layer computes the data, while the batch normalization regularizes the output, this way the next dense layer can capitalize on a normalized input improving stability and performance. The dropout layer deactivates some of the nodes of the next layer promoting a reduction in overfitting and a more resilient network. The batch normalization usually reduces (or sometimes prevents) the need for a dropout layer, but since our network suffers from a high risk of overfitting it was applied, nonetheless.

The order of operations in this block is crucial. Sometimes the batch normalization layer is applied after the computations of the dense layer but before the activation function; however, experimental evidence suggests that putting the batch normalization layer after the activation function may slightly improve performance [77]. It is also more theoretically grounded: applying the batch normalization layer after the activation function guarantees that the next dense layer will have an input with zero mean and unit variance, something that is not guaranteed if the opposite order of operations is performed, especially because the activation functions are not linear.

The dropout layer must be placed after the batch normalization layer. This is because if it is put before the batch normalization layer it may change the statistical distribution of the data since it is blocking part of the flow of information, disrupting the input data that the next layer will receive.

If a residual connection is present, it will take the place of the dropout layer to prevent the same problem. The purpose of a dropout layer is to partially block the flow of information in the network. If the dropout layer is put after the residual connection, it will change the statistical distribution of the data because it blocks part of the data, but if it is put before the residual connection the effect may be even more disruptive. A dropout layer put before a residual connection will block part of the most recent data not allowing them to be summed to the older data. This not only will change the statistical distribution but will make indirectly the older data more predominant because the previous layer has limited influence due to the effects of the dropout. Additionally, the random selection of the blocked data may cause further instability because the next dense layer has no way to reliably assess whether it is receiving a sum of both the data types or only the older one.

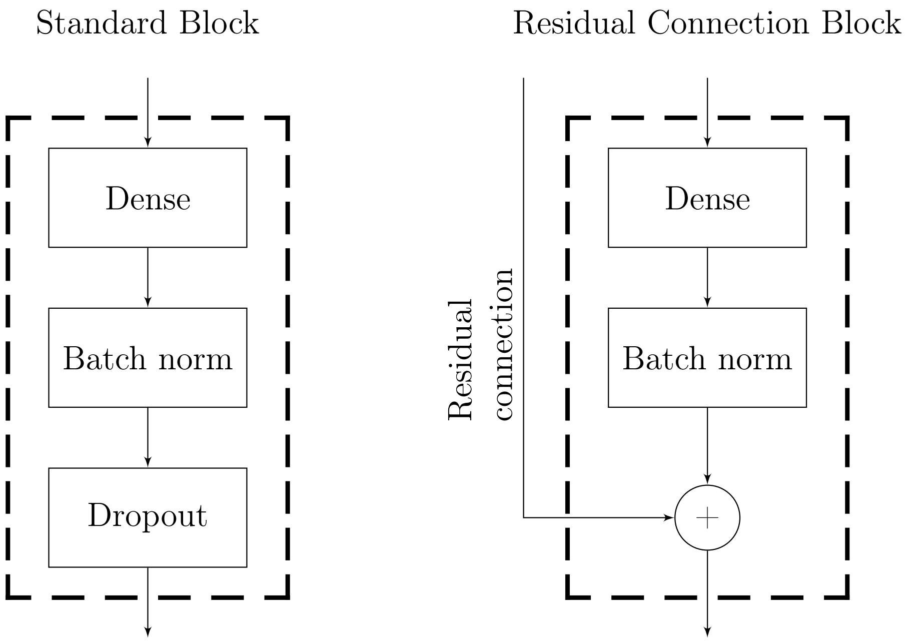


Figure 2.4: A diagram of the block containing a dense layer, a dropout layer and a batch normalization layer, the variant with residual connections is also present.

2.3 Main Net

This network, in particular, is very complex and its behavior may be unpredictable if tuned in the wrong way. Most of the problems of this network were tied to the polarization of its response and the way to mitigate the long time required to tune and train the model. This specific kind of network is not well documented, so we relied on heuristics and a big part of the decision-making process was based on trial and error.

Loss function

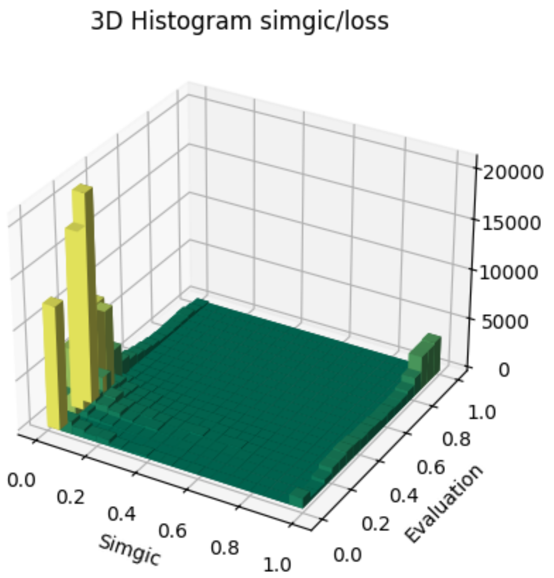
A lot of loss functions were tried for the Main Net, the MAE (Mean Absolute Error), the MSE (Mean Square Error) and even a custom loss based on a 3D representation of the desired output (see the subchapter 2.5 for further details). All these functions provided a very polarized response from the network, causing almost all the predictions to fall in one of the corners of the boundaries (most of the values are very close to 0 or to 1, in both axes). This sharp response was undesirable: it undercut the use of a decimal output instead of a categorical one because it provided almost no nuance to the prediction. Additionally, thresholding was impossible to use causing an overall dip in the quality of the prediction, even if the overall numerical error was smaller.

The loss function of choice became the RMSE (Root Mean Square Error), and is defined by this formula:

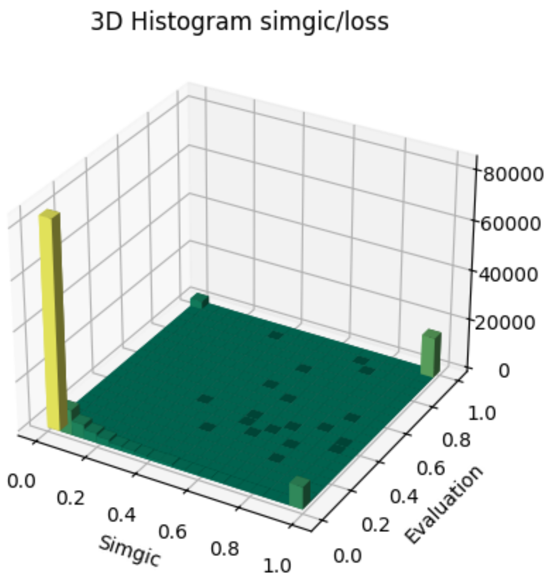
$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

- n: number of samples.
- y_i: true simgic, obtained by the comparison between Argot's prediction and the function obtained from the ground truth.
- $\hat{y}_i$: predicted simgic obtained from the Main Net.

Since the loss is between 0 and 1 (because the two simgics can reach values between 0 and 1), the bigger losses (close to 1) will be adequately penalized while the smaller errors will only be slightly penalized. RMSE also provides a more balanced error distribution leading to a better distribution of predictions.



(a) Distribution of the predictions of a Main Net prototype with RMSE as the loss function on a test dataset



(b) Distribution of the predictions of a Main Net prototype with MSE as the loss function on a test dataset

Figure 2.5: At first glance it may seem that the model with RMSE loss function (Figure 2.5(a)) is the inferior version. Even if the model with MSE loss function (Figure 2.5(b)) has an overall lower error of 0.1522 against the 0.1821 of the model that uses RMSE, it has a lower performance. This is because the most polarized response can't be thresholded and is most likely overfitted. A gentler response causes a more nuanced prediction, and it is easier to catch possible errors.

Loss Function	Cel Components	Mol Functions	Bio Processes
RMSE	$F1_{RMSE} = 0.78$	$F1_{RMSE} = 0.67$	$F1_{RMSE} = 0.71$
MSE	$F1_{MSE} = 0.76$	$F1_{MSE} = 0.64$	$F1_{MSE} = 0.69$

Table 2.1: A comparison of the results achieved by both the loss functions.

2.3.1 Training process

To explore all the possible parameter configurations, we used the Hyperband algorithm during hyperparameter tuning. The Hyperband algorithm is an extension of the successive halving algorithm that uses adaptively allocated resources to find the best configuration [78]. The successive halving algorithm tries to address the problem of resource allotment between several possible combinations n and a finite budget B, providing each combination with the average amount of B/n resources. If we want to test a lot of n the B/n will be very low so it will be possible to test a small number of epochs, conversely if we settle for a small n it will be possible to test those combinations further, this is very beneficial because the initial performance of a combination may be misleading, providing with better loss down the line than an immediately more promising configuration. The parameter η can be used to regulate the aggressiveness of the halving, during tuning we set it to $\eta = 4$, as advised in the original paper [78].

Instead of naively allocating B/n resources per combination, the Hyperband algorithm allocates the necessary resources required to distinguish the performance of each configuration. After a set number of epochs, the combinations are halved by deleting the worst-performing ones, this way the resources can be used to further research more promising configurations. The Hyperband algorithm is very fast and efficient, providing $\times 20$ faster times than a random search and drastically improving the tuning process [78].

When the best model has been found, we used those hyperparameters to train the definitive model with the full dataset. The training used k-fold validation with $k = 6$ and a test set of 15% of the dataset.

2.4 Future Net

The main problem that emerged during training was the unbalanced dataset. In this dataset, the majority class 0 (the regular predictions) is 82.7% of the entire dataset, while the minority class 1 (the predictions that anticipate the experimental evidence) is only the 17.3% of the dataset. The unbalanced dataset caused bad performances and instability during training, so most of the efforts were directed at tackling this problem and obtaining stable and good performances.

Adaptive class weighting

To mitigate the problem of the unbalanced categories in the dataset an adjusted class weighting system was implemented. When a dataset is imbalanced, if the two classes have the same weight, the network may opt to completely ignore the minority class and predict the majority class no matter what, because it has a small incentive to learn how to discriminate one from the other. The adjustment of the learning weight forces the network to consider the minority class to not incur an undesirable big loss. The loss value must be carefully tuned because if the weight is too high, the opposite problem may happen and cause the network to only predict the minority class.

The conventional method to decide the balanced weight of the classes is:

$$W_i = \frac{n_{tot}}{n_{classes} \cdot n_i}$$

- W_i: weight for class i.
- n_{tot}: the total number of samples.
- n_i: the number of samples belonging to class i.

Since class 1 is $\approx 17.3\%$ of the total dataset it should be 4.81 (if class 0 has a normalized weight of 1) but the empirical results suggest that a split of 4 and 1 leads to better performance. This is unsurprising because with oversampling part of the separation between classes has been mended, making the weights 4 and 1 higher than expected by the function. A higher weight also helps achieve a higher recall (rather than a high precision), a metric that is more interesting for us. Being an explorative tool it is more interesting if we can predict consistently more anticipatory labels, even at the cost of some more false positives. The nature of the 0 class also helps, because due to the missing information problem, we can't exclude that any 0 may be a 1 instead, allowing the network to express itself and find more anticipatory predictions.

Loss function

The loss function used for the Future Net is the Balanced Cross Entropy. The Binary Cross Entropy (previously seen in equation (1.4)) is the typical choice for binary class prediction, so it was used as a starting point. To address the unbalanced dataset and improve performance we transitioned to a Balanced Cross Entropy that uses our set of custom weights as coefficients. The Balanced Cross Entropy differs from the Binary Cross entropy because it multiplies the loss of each sample by a coefficient that changes with the class. This implementation achieved very good results. The resulting loss function is the following:

$$\mathcal{L}_{BCE} = -\frac{1}{n} \sum_{i=1}^{n} \left[\alpha_1 \cdot y_i \cdot \log(\hat{y}_i) + \alpha_0 \cdot (1 - y_i) \cdot \log(1 - \hat{y}_i) \right]$$

In this equation, α_0 is the coefficient that multiplies the loss for class 0, while α_1 does the same for class. The value of these coefficients will be determined during the tuning process.

During the tuning process, the Focal Loss was also tested. The Focal Loss is a more complex version of the Binary Cross Entropy that can differentiate between easy and hard-to-predict examples. This property allows the loss to improve the contribution made to the training from hard examples and reduce the impact of easy examples [79]. Overall, the high quantity of parameters that required tuning and the inconsistent improvements to the performance of the network convinced us to use the more dependable Balanced Cross Entropy.

2.4.1 Training process

To tune the hyperparameters we used a custom random tuner. The random tuner is a rudimentary and inefficient algorithm that tunes parameters simply by trying different combinations and recording which are the best ones. The random tuner does not use the efficient techniques implemented in the Hyperband algorithm, but it requires setting preemptively the train and validation datasets to train on it. Since we used the tuning process to search for the best oversampler the implementation of the Hyperband algorithm would have increased greatly the complexity of the code, so it was not deemed a worthy addition and we kept using the random tuner. Due to this limitation, it wasn't possible to implement k-fold validation during the tuning process, so to keep the conditions consistent and because it was easier to train than the Main Net this technique was not implemented.

To evaluate the performance of the Future Net during training and tuning we did not use the validation loss. For this network, the validation loss was not a good predictor of performance because the best loss was often associated with overfitting. To avoid this problem the F1 score evaluation of the network was directly implemented in the training and tuning process.

The F1 score (equation 1.3) is a metric used to summarize the performance of a network, joining the precision and the recall in a single metric. When we refer to the F1 score we are considering the one respective to the minority class 1, since this is the more interesting and the less performing one.

After each epoch, an F1 score evaluation was run using the validation set, this was the metric used to trigger the early stopping and to evaluate each run of the random tuner. The computation of the F1 score slowed both the training and the tuning process because it required the evaluation of all the validation datasets from the network each epoch. After the computation of the predictions, an iterative thresholding process was started to find the best threshold. Each number between 0 and 1 (with a step of 0.02 during validation and 0.005 during testing) was tested and the value that provided the best results was chosen.

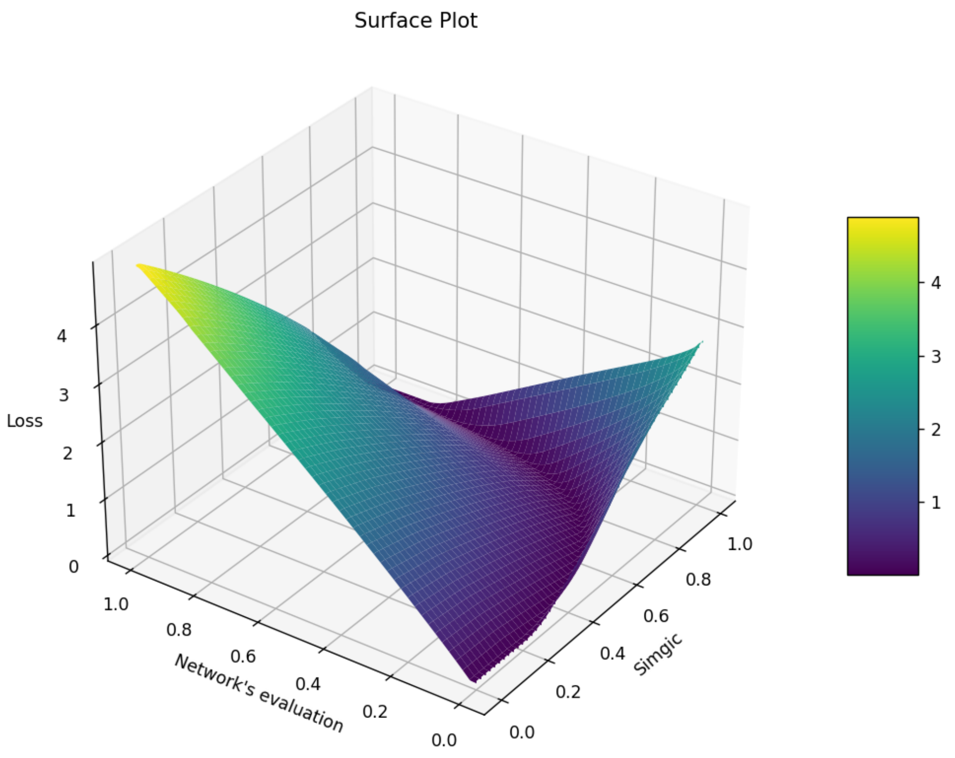
This also allowed us to monitor in real-time the development of the threshold and of the F1 score.

2.5 Failed approaches

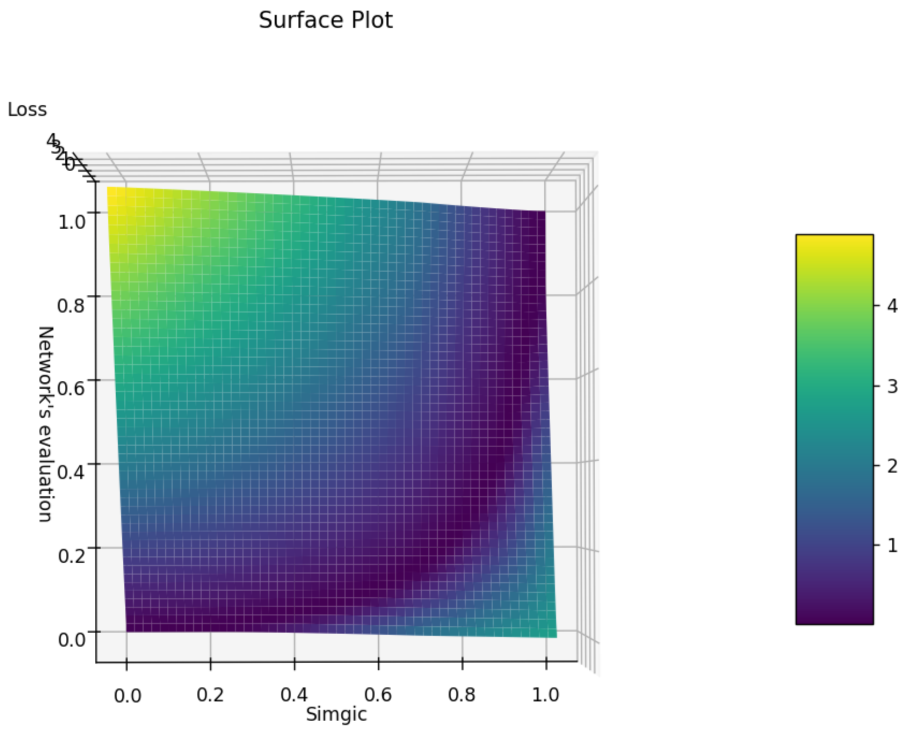
A lot of techniques were unsuccessful in improving the NNs. In this subsection, we will illustrate what didn't work as expected and try to explain the possible reason behind the lack of improvements.

2.5.1 Custom Loss for the Main Net

At the start of the project, the simgic was not the target value, instead, we used a custom loss aligned to the desired evaluation for each simgic. The evaluation metric was designed to evaluate the quality of the Argot prediction without being a one-to-one recreation of the simgic metric. To achieve the desired shape the function was very complex and was passed through Gaussian filters to smooth it out and simplify the job for the SGD algorithm. The two variables of the function were the network's evaluation and the simgic from the ground truth. To compute the loss in a reasonable time we calculated the values of the function on a grid and used interpolation to find an approximation of the loss value.



(a) Side view of the custom loss function.



(b) View from above of the custom loss function

Figure 2.6: This is a graphical depiction of the custom loss. It is possible to see that the network's evaluation and the simgic are not linearly related, because only a high simgic can be considered reliable enough for a good evaluation.

This custom loss was discarded for multiple reasons:

- The high computational complexity slowed down the training process considerably. The density of the interpolation grid controlled the computational complexity, so if the training process required a more precise response, it would exacerbate the computational requirements, increasing the time required to train and tune the neural network.
- This loss function enhanced the problems regarding the highly polarized dataset, encouraging the network to always produce extreme evaluations of 1 or 0, with no intermediate values.
- The shape of the loss was completely discretionary, with no validation in the literature. This meant that the behavior of the network would be unpredictable and hard to understand. This problem complicated the development process because even the most common and simple performance-enhancing technique would make the network behave erratically when implemented.

In the end, it was simpler, faster, more explainable and better to use the simgic as the target variable and use a conventional loss function like MAE or RMSE.

2.5.2 Custom categorical multiplier for the Main Net

Before the implementation of a resampling method, but after switching to a conventional loss function, a custom categorical multiplier was implemented into the loss function to limit the bias in favor of the majority class in the Main Net. The Argot predictions fed to the Main Net can be roughly subdivided into two categories: good predictions (simgics >0.75, the minority class) and bad predictions (simgics <0.75, the majority class). The Main Net usually has better performance evaluating accurately bad predictions rather than good predictions, this is due to the imbalance in the dataset that favored the majority class.

The simgics value of the predictions (our target value) is a number between 0 and 1, but, due to the high polarization of the dataset, most of the data is close to 0 or 1, with a negligible portion of the data falling into an intermediate value. This means that the dataset could be roughly categorized into bad and good predictions even if this was not, explicitly, a categorization problem. To penalize further the evaluation errors made to the minority class a check was implemented. This check would multiply (with a value higher than 1) the losses related specifically to the predictions with a simgics>0.75 to increase the loss for that particular evaluation.

This approach didn't work because it exacerbated the polarization, pushing the predictions further from the middle. This effect caused a worse thresholding process and an overall slight decline in performance. So, we switched to more conventional methods to rebalance the network response, like rebalancing the values of the dataset via under/oversampling.

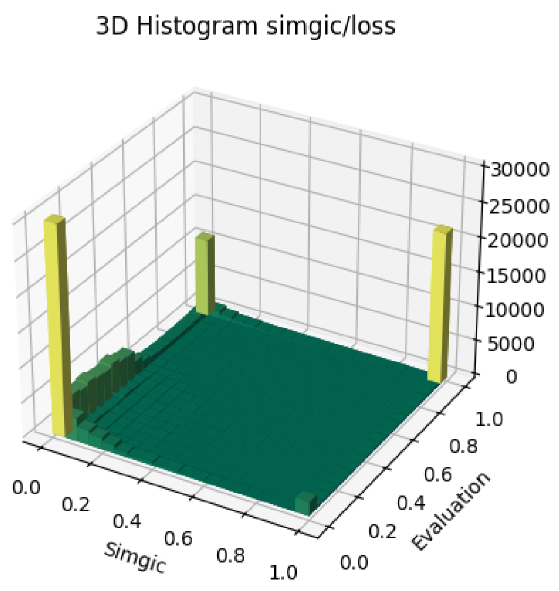


Figure 2.7: This is the behavior of one of the most advanced prototypes that used the custom multiplier. We can see that it has a strong tendency to misclassify data with a low simgic as it had a high simgic.

Performance of the prototype with multiplier				
Cellular components				
Class	Precision	Recall	F1 score	Support
0	0.96	0.94	0.95	40016
1	0.73	0.81	0.77	8607
Threshold				
0.98				
Molecular functions				
Class	Precision	Recall	F1 score	Support
0	0.94	0.90	0.92	58490
1	0.61	0.73	0.66	12443
Threshold				
0.94				
Biological processes				
Class	Precision	Recall	F1 score	Support
0	0.93	0.88	0.90	23650
1	0.65	0.78	0.71	6794
Threshold				
0.98				

Table 2.2: The performance of the model. During prototyping, we used a smaller dataset (only 1000000 entries, so the test set that produced this benchmark contained 150000 entries) because it allowed for faster development iterations, so the support for each class and ontology was reduced. The multiplier for this particular model was multiplier= 5.

Training this kind of network was very complex, it was very prone to overfitting and to reliably assess the performance of the network the validation loss could not be trusted because often, a better loss meant only more overfitting. To evaluate the network, we were required to analyze the behavior of the output and the test classification statistics, making the tuning process very complex. With more time this method could, potentially, be explored further with success but during the development, we could not improve the network by using it. The thresholding was also almost useless, due to the highly polarized response the network would often classify the data using 0 or 1 almost every time. The evidence for this behavior can be found in the preferred threshold for each ontology since all of them are very close to 1.

Chapter 3

Results and discussion

In this chapter, we will analyze in detail the structure of the Main Net and of the Future Net obtained after the tuning process. The performance of both networks will also be discussed, providing graphs about their behavior.

3.0.1 Use of total score

The total score has been very heuristically useful during the development of Argot 2.5 because it represented a good approximation of the reliability of the prediction, but its behavior is not completely clear. Due to its opacity, there have been efforts to exclude the total score from the input data, but it proved to be too useful. At the start, we tried to not use it to make more theoretically sound networks but the overall lack of data in each entry and its interpretation compelled us to use it as a metric, nonetheless. The addition of the total score dramatically improves performance in both the Main Net and the Future Net, so in this chapter, we will consider the best-performing models that use it as an input metric.

3.1 Model 11

To find the best architecture for our neural networks we checked 13 different structures. After the tuning process, we found that the best model architecture for both the Main Net and the Future Net is Model 11, even if the tuning process was independent between the two networks. Model 11 is an architecture based on residual connections that uses the block system shown in Figure 2.4. A block can be a standard block or a residual connection block, a group of blocks is a structure made of two standard blocks and a residual connection block stacked in this order. There are 17 different groups of blocks made of two standard blocks (light orange, as shown in Figure 3.1) and one residual connections block (dark orange, as shown in Figure 3.1), except for the last one which is a simple dense layer of width 1 (effectively a single neuron, in light blue, as shown in Figure 3.1) that uses the sigmoid function to compute the output. The first 14 groups of blocks are all of width 7 so they can share residual connections without any problem, but the last 3 groups have a different width so they can't use the previously established residual connections.

Each group of blocks is influenced by the previous one via a residual connection. In addition to that, every odd group of blocks (starting with the first) has its residual connection propagated throughout the network. This pattern can't continue through the last 3 groups because they have different widths than the rest of the network.

This model has 49 different dense layers and 3073 trainable parameters. Due to the high number of layers, we would expect a higher number of trainable parameters, but almost all of them have a width of 7. Compared to the other architectures we tested, this one is quite deep but is one of the only four structures with a constant width because a lot of the tested architectures had variable layer widths. The residual connections of this model are also quite simple. Due to the computational cheapness of residual connections, they can often be used to increase the complexity of a model without the addition of further layers, so some of the tested models had very intricate residual connection patterns. Interestingly, this is one of the architectures with the most regular patterns and an overall low number of independent residual connections.

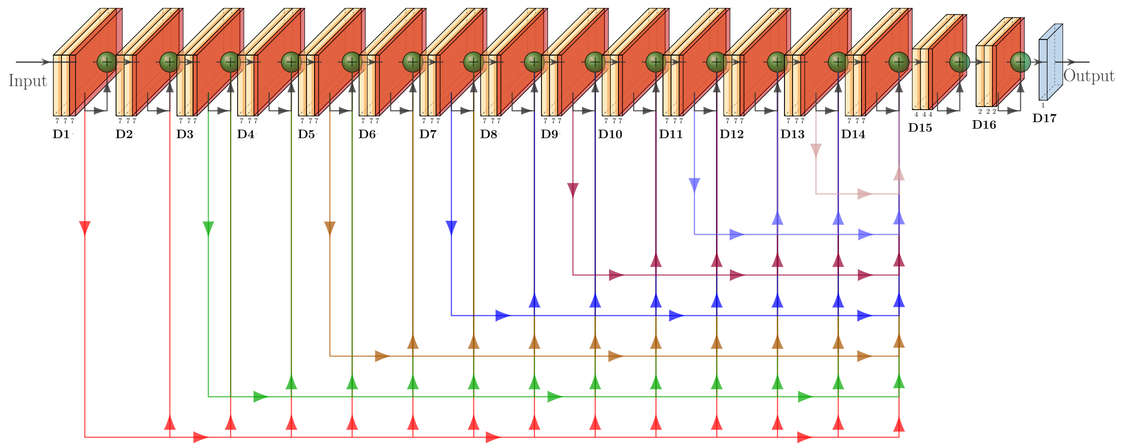


Figure 3.1: This is the architecture of Model 11. A detailed description of its depiction can be found in the Appendix and a bigger picture can be seen in Figure 4.1.

3.2 Main Net

In this section, we will evaluate the performance and the behavior of the Main Net and compare it with the previous method used to rank the predictions of Argot.

3.2.1 Choice of metrics

To evaluate the Main Net, we focused on two kinds of metrics. At first, we used the distance between the true simgic, the evaluation of Argot's predictions compared to the functions in the ground truth, and the predicted simgic obtained from the Main Net[1].

Then we added the classification between good and bad predictions, to obtain a more complete evaluation.

The obvious approach to evaluate a prediction is computing the difference between the real metric and its prediction, but basing the choice of the best model

[1]E.g. a prediction from Argot is compared with the function in the ground truth producing the simgic that encapsulates their similarity $simgic_{true}$. Then we compute the simgic from the Main Net, obtaining $simgic_{MN}$. To compare these two values we calculate the absolute difference $abs\ error = |simgic_{true} - simgic_{MN}|$ and use it as a metric that can quantify the error of the model.

purely on this value can lead to problems. The use of an average hides the distribution of the values, hence it is not enough to determine if the prediction had a consistent error or was prone to spikes. These two scenarios may change the true performance drastically; for our purposes, a consistently small error is almost negligible, while a consistently smaller error with occasional spikes is more problematic. The reasons are twofold:

- A consistent error makes the quality of the evaluations more predictable. This means that we can reasonably expect each prediction to be around its true value, if this is not true the predictions become less informative because they can occasionally be completely wrong.
- For our use case we are particularly interested in a prediction if it has a value close to 1 (generally if it is >0.75). This means that if a prediction with low simgics has a consistently small error it can be safely considered a bad prediction from Argot, while if the same prediction has a small probability of being completely wrong it may disrupt our evaluations.

The Main Net is not explicitly a classifier, but it can be used to detect good or bad Argot predictions via thresholding. This is useful information because it helps us in selecting the best predictions to consider. To evaluate this type of classification we used the F1 score to summarize the performance of the network for each class, the F1 score of the good Argot predictions was the target metric that was maximized during the thresholding process, after the selection of the optimal threshold. To provide a wider spectrum of information we also added precision, recall and accuracy.

3.2.2 Resulting network

After the tuning process, the model that performed better had the following hyperparameters:

Tuned network	
Hyperparameter	Value
Batch size	50
Dropout	0.25
Leaky alpha	0.15
L1 regularizer	0.005
L2 regularizer	0.005
Init distribution	uniform
Resample type	None
Optimizer	Adam
Initial learning rate	0.01
Decay steps	5000
Decay rate	0.98
Last layer bias	False

Table 3.1: The values of the hyperparameters in the Main Net after the tuning process.

A brief description of the hyperparameters listed in the table:

- Batch size: the amount of single entries for each batch.
- Dropout: the ratio of deactivated neurons.
- Leaky alpha: the value of the parameter that controls the negative slope in the leaky ReLU.
- L1 regularizer: the value of the parameter that controls L1 regularization.
- L2 regularizer: the value of the parameter that controls L2 regularization.
- Init distribution: the kind of distribution that generated the starting weights for the leaky ReLU or ReLU neurons, the He uniform distribution or the He normal distribution.
- Resample type: which type of resample techniques were used, oversampling, undersampling, both of them or neither of them.
- Optimizer: which kind of optimizer was implemented between SGD, SGD with momentum, SGD with Nesterov momentum, Adadelta or Adam.
- Initial learning rate: the starting learning rate.
- Decay steps: the number of steps required to trigger the reduction of the dynamic learning rate.
- Decay rate: the amount of the learning rate that is kept after each iteration of the reduction of the dynamic learning rate
- Last layer bias: if the last layer uses bias, or if it is set to 0.

As expected, the best optimizer is the Adam optimizer. Overall, the Adam optimizer is very powerful and reliable, requiring little tuning to achieve very good performances and reach convergence quickly.

The Main Net was trained in 17 epochs while having a patience of 15, where the patience is the number of epochs without improvement necessary to stop the training. This means that the Main Net stopped improving during training after only the third epoch because it converged rapidly. This is a desired behavior because it means that the model is fit to solve the problem and easily understands the data. The overall low complexity of the dataset and the use of the Adam optimizer (which is known to hasten the convergence of its models) may also have influenced the result. This behavior is consistent with the one that we witnessed during the tuning process.

3.2.3 Performance

The performance of the network was computed by classifying the simgic obtained by comparing Argot's predictions and the function contained in the ground truth in two different classes. Each Argot's prediction can be considered good if it is close to one of the functions contained in the ground truth, for the same gene product. However, if the prediction bears no correspondence (or if it is too weak to be significant and useful) it will be considered a bad prediction. This classification is made with a cut-off threshold of simgics=0.75. If the prediction has a simgics>0.75 it is considered a good prediction (class 1), but if the simgics<0.75 it is considered a bad prediction (class 0). To evaluate the Main Net, we computed its predictions and found the best threshold that would cause the best classification. The metric that was maximized during this process was the F1 score of class 1. The following tables contain the performance of the Main Net computed on the test set.

Test dataset				
Class	Precision	Recall	F1 score	Support
0	0.94	0.94	0.94	2147126
1	0.72	0.75	0.73	464190
Overall accuracy				
0.90				

Table 3.2: The performance of the Main Net during testing. Class 0 refers to bad predictions (simgics<0.75), while class 1 refers to good predictions (simgics>0.75).

Comparison with the alternative

To be a success, the Main Net should be an improvement on the previous method used to rank the predictions of Argot 2.5. The previous method implements a spline that scales solely on the total score. The spline uses the total score to compute a value between 0 and 1 to evaluate the quality of the network and ignores any other output from Argot. The behavior of the spline can be seen in Figure 3.2.

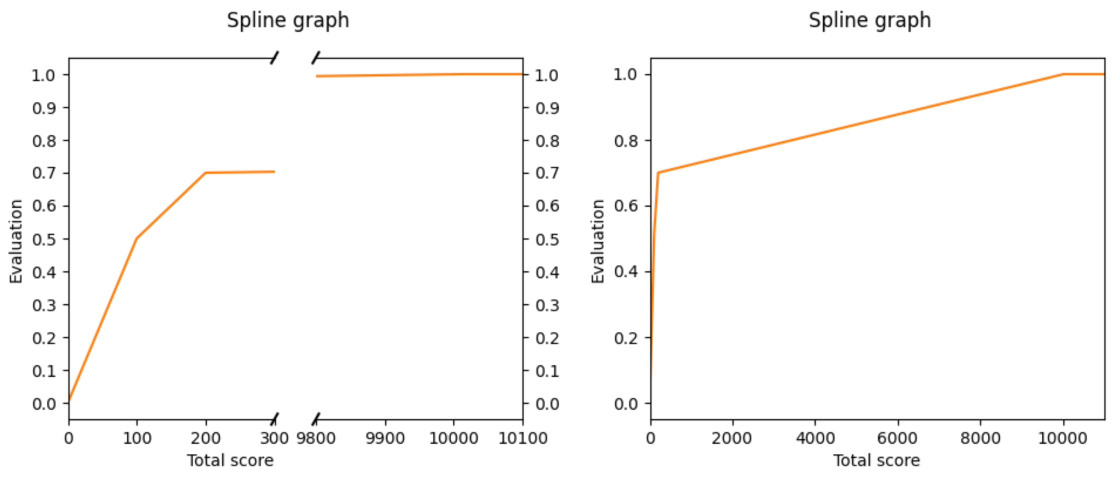


Figure 3.2: Two representations of the behavior of the spline for all the possible total scores. The one on the left skips the long slope between the total scores $[200, 10000]$ that goes from the evaluation of 0.7 to 1.0.

To verify if the Main Net is an improvement over the spline the performance of the two methods are compared on the same dataset, the test set of the Main Net.

The evaluation distribution and the absolute prediction error are evaluated first. In Figure 3.3 the simgics of Argot's predictions (blue), the evaluation of the spline (green) and the evaluation of the Main Net (orange) are depicted. As we can see, the predictions of the Main Net are generally slightly higher than the ground truth, while the spline has an unexpected spike of around 0.7 for each ontology.

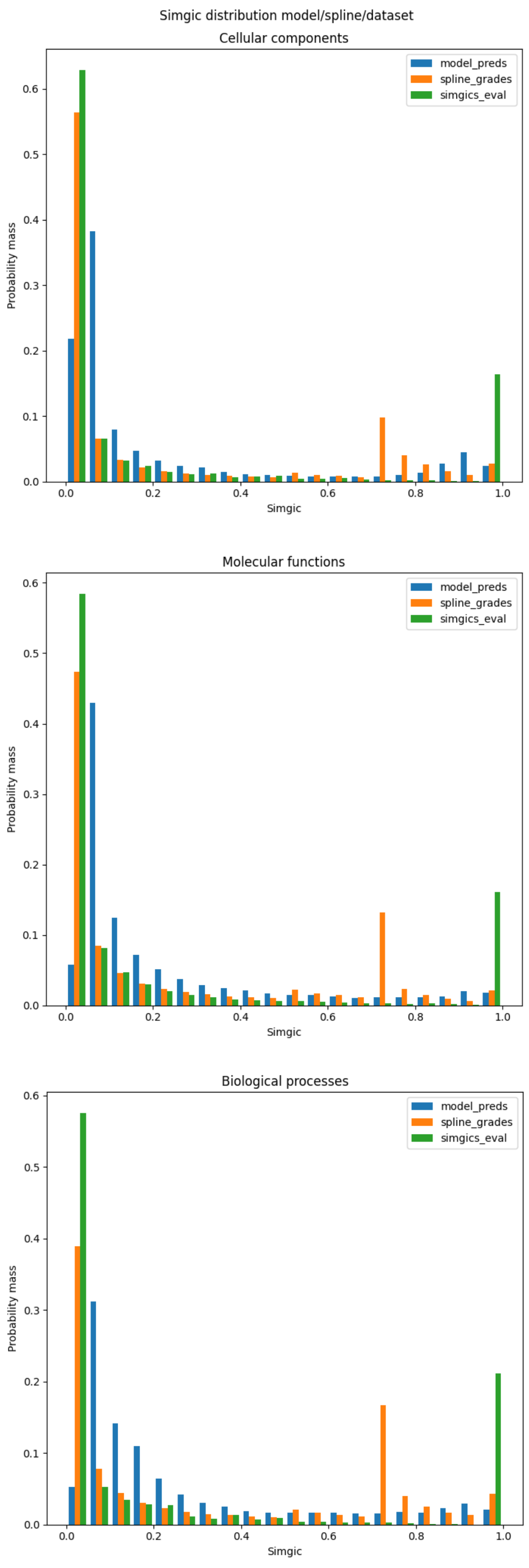


Figure 3.3: A representation of the distribution of the prediction of the Main Net (blue), the spline (orange) and the ground truth (green).

In the following Figure 3.4 we can see that even if the average absolute prediction errors are close not only the Main Net is consistently better, but it has a more consistent low error rather than the spikes of the spline.

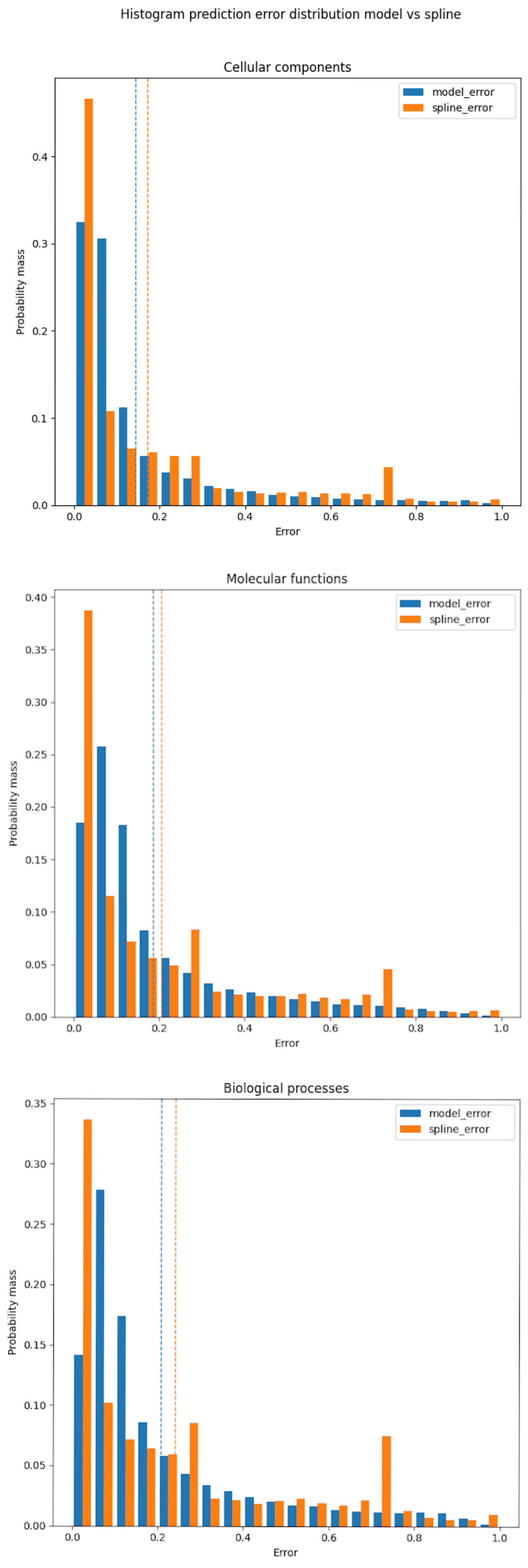


Figure 3.4: A representation of the absolute error of the prediction for both the Main Net (blue) and the spline (orange).

The most important metrics to evaluate are the classification metrics, how well can both these methods recognize good Argot predictions (class 1, simgics>0.75) from bad Argot predictions (class 0, simgics<0.75). To provide the maximum performance from both the spline and the Main Net each ontology has been thresholded separately, as seen in the following table.

Classification prediction							
Cellular Components							
Spline				Main Net			
Class	Precision	Recall	F1 score	Class	Precision	Recall	F1 score
0	0.95	0.97	0.96	0	0.96	0.96	0.96
1	0.82	0.73	0.77	1	0.81	0.78	0.80
Accuracy		Threshold		Accuracy		Threshold	
0.93		0.72		0.93		0.48	
Molecular functions							
Spline				Main Net			
Class	Precision	Recall	F1 score	Class	Precision	Recall	F1 score
0	0.94	0.90	0.92	0	0.94	0.93	0.94
1	0.60	0.73	0.66	1	0.67	0.73	0.70
Accuracy		Threshold		Accuracy		Threshold	
0.87		0.70		0.89		0.42	
Biological processes							
Spline				Main Net			
Class	Precision	Recall	F1 score	Class	Precision	Recall	F1 score
0	0.91	0.94	0.92	0	0.93	0.91	0.92
1	0.73	0.65	0.69	1	0.69	0.75	0.72
Accuracy		Threshold		Accuracy		Threshold	
0.87		0.72		0.88		0.40	

Table 3.3: A comprehensive description of the performance of both the Main Net and the spline for each ontology.

Due to the lack of data for each entry, the Main Net could not perform at the desired level because neural networks, notably, perform better with data points that have more features (considering datasets with the same number of entries). This problem persisted even when we chose the type of network that was less reliant on this type of input data, and unfortunately, the high quantity of data was not enough to fully close the gap. Even after experiencing these problems, the Main Net has a small edge in performance due to the fact that it can capitalize on multiple sources of data (all the Argot outputs) while the spline uses only the total score. Although the improvement is small, it is still important because during the CAFA competitions most of the AFPs are really close in terms of performance, and even a small increase in the F1 score may strongly impact the final ranking of the method. Additionally, due to the continuous developments of Argot, the margin for improvement is reduced and even smaller upgrades become progressively harder. Taking this information into consideration, we can consider the Main Net a successful improvement on the spline.

3.2.4 Analysis on quality

We analyzed the behavior of the Main Net on all the data marked by the ground truth, to analyze if the results changed between data of different origins. These datasets will contain only Argot predictions that match with functions of the ground truth, this means that they contain only good predictions (with simgic>0.75). The expected result was an improved performance regarding the experimental data, because it is more reliable, and a worse performance for electronically extrapolated data. Surprisingly, the network evaluated slightly better the electronic data.

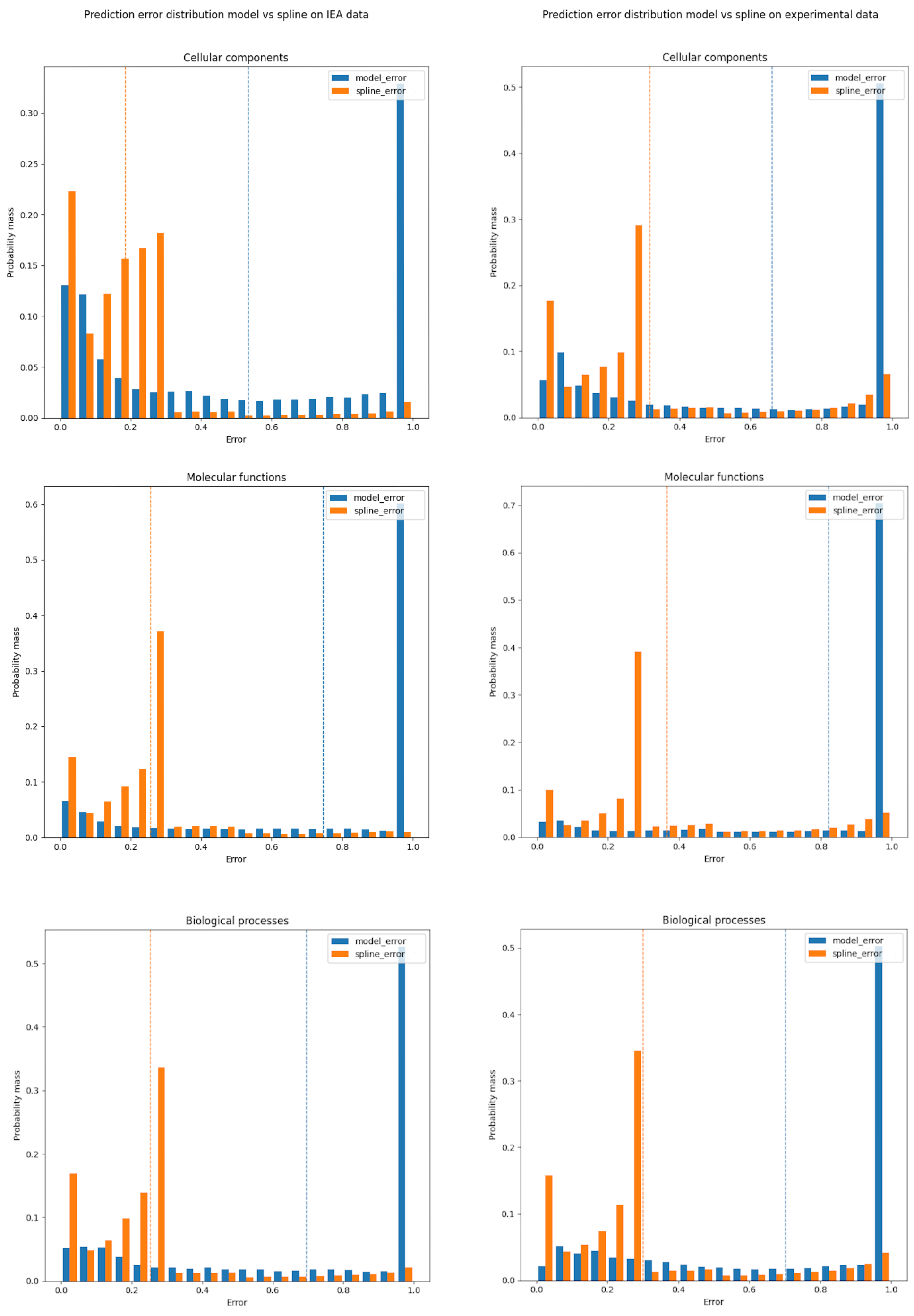


Figure 3.5: This is the distribution of the absolute evaluation error for both types of data: electronic (left column) and experimental (right column). The dashed lines are the averages, the blue histogram represents the evaluation error of the Main Net, while the orange histogram represents the performance of the spline.

As we can see in the graphs in Figure 3.5 it may be more evident in some ontologies, but the classification is better for the electronic data.

We devised two main explanations for this apparent incongruity.

First, even if the electronically extrapolated data is generally less reliable, it is still of sufficient quality to perform adequately with the Main Net.

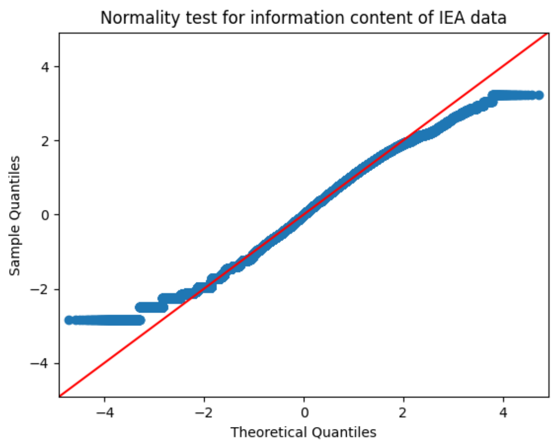
Second, the experimental data may be intrinsically different from the electronic one. The experimental data may be very specific (deep into the GO DAG) while the electronic data tends to be less specific (shallower into the GO DAG). If that were the case, the experimental data (even if close to 50% of the whole ground truth) may represent annotations that are rarer and more difficult to interpret than the more commonly present annotations of the electronic data.

To corroborate this hypothesis, we checked the distribution of the information content in the two types of data. The information content is a metric that is directly correlated to the specificity of the annotation. The results of the analysis of the distribution are in the following table.

Distribution of the information content				
Data type	Average	Median	Std. deviation	Support
IEA	11.15	11.11	3.75	885276
Experimental	12.50	12.64	3.50	2124938

Table 3.4: A description of the distribution of the information content present in both the fully electronic (without UniProtKB search) and the experimental data.

From the data, it is already possible to see that there are some striking differences in the average and the standard deviation but to be certain, we performed a two-tailed Welch's t-test [80]. The Welch t-test is a statistical test that checks if two groups of data are sampled from the same population, it is more robust than the conventional t-test in the case of two groups of data with different variances, such as this one. The choice of the Welch's t-test was informed by the difference in variance ($std\ dev^2$), in this case, a standard t-test would not have been appropriate because the difference in variance would have led to an underestimation of the p-value [81]. This test requires that both of the distributions must be normal; to verify this assumption we used a quartile-quartile plot (or q-q plot). The q-q plot is a graphical representation of the quantiles of the analyzed cumulative distribution confronted with the normal cumulative distribution, if both the curves follow the same behavior we can assume that the tested distribution is normal. As seen in Figure 3.6 both the distribution of the information content type 0 and type 2 have a normal q-q plot, this means that they can be considered sampled from a normal distribution and that we can finally apply the Welch's t-test.



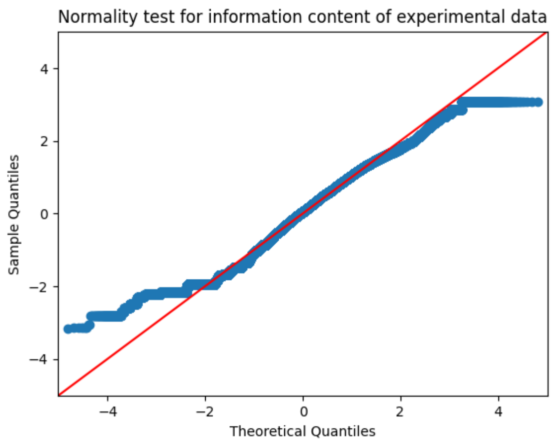


Figure 3.6: This is the q-q plot of both types of information content. The information content cumulative distribution (blue dots) has a similar shape as the cumulative normal distribution (red line). This means that they have a normal distribution.

The p-value of the test was $p = 0.0$[2], so the null hypothesis is rejected, and we can assume that the distributions are significantly different. This means that our hypothesis has ground to stand on and is plausible.

Another surprising property that is apparent from the graph in Figure 3.5 is the low performance achieved by the Main Net in these conditions. To explain it, we must consider that this special dataset contains only good Argot predictions, this is because we selected all the Argot predictions that represented functions present in the ground truth.

The structure of the dataset brought three problems that negatively influenced the performance of the Main Net:

- The Main Net was tuned and trained using a dataset that had only a minority of good Argot predictions, leading the network to recognize only a proportion of good predictions similar to the one present in the conventional datasets (dataset 1M and the full dataset). As we can see in Figure 3.7 the proportion of predicted good predictions over all the predictions is similar between the two datasets despite the different distribution.
- Due to the lack of bad predictions, the standard scaling performed during the data preprocessing procedure created a radically different distribution. This unexpected distribution may have affected negatively the predictions of the Main Net.
- This new distribution may also have caused problems in the batch norm layers of the network. These layers were trained to expect and process a different distribution. This discrepancy may have caused problems in all the 49 different batch norms layers propagating the error through the network.

Considering these adverse conditions is not surprising that the Main Net achieved such a low performance.

[2]Technically, the p-value can never be zero but due to technical limitations the software computed this value. This is because the calculated p-value was a number smaller than the smallest number representable by a float64 data type (it was $< 2.23 \cdot 10^{-308}$), the test statistic was approximately ≈ -272.7616 if the reader wants to check the p-value independently.

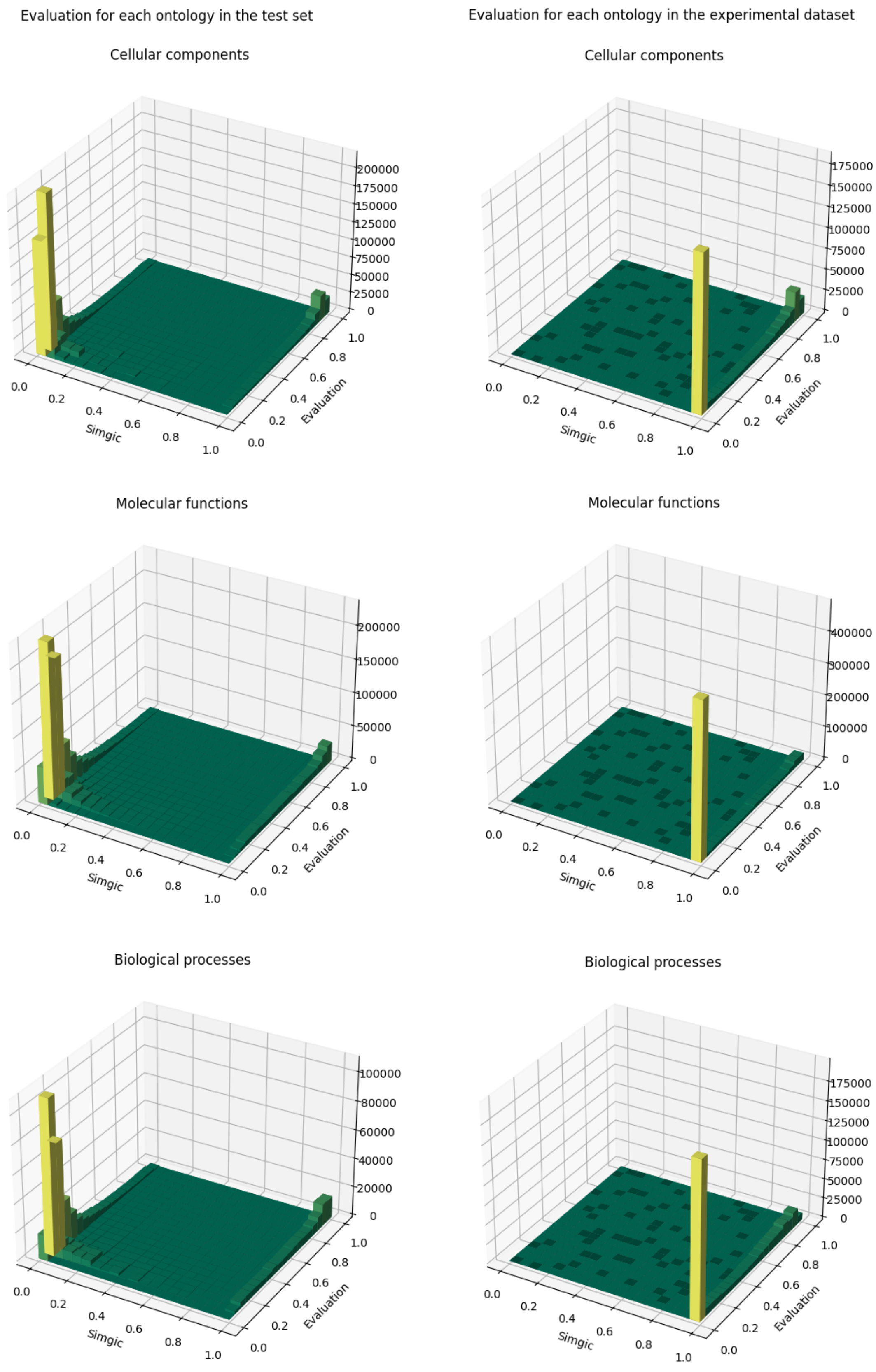


Figure 3.7: This is a comparison between the test dataset (left column) and the experimental dataset (right column). As we can see the distribution of the Main Net prediction is similar despite the different distribution of the two datasets.

3.3 Future Net

In this section, we will evaluate the performance and the behavior of the Future Net.

3.3.1 Choice of metrics

To evaluate the performance, we preferred to focus on the F1 score during training and tuning because it is a good summary of the overall performance of a network for each class. Refer to equation 1.3 to see how it is calculated. We also computed precision, recall and accuracy. The accuracy is useful because it tells us the proportion of correct predictions over all the predictions.

3.3.2 Resulting network

After the tuning process, the model that performed better had the following hyperparameters:

Tuned network	
Hyperparameter	Value
Batch size	100
Dropout	0.2
Leaky alpha	0.0
L1 regularizer	0.01
L2 regularizer	0.01
Init distribution	uniform
Class weights	0: 1.0, 1: 4.0
Resample type	SMOTE
Oversampling strategy	0.9
K neighbors	20
Optimizer	Adam
Initial learning rate	0.01
Decay steps	250
Decay rate	0.98
Last layer bias	True

Table 3.5: The values of the hyperparameters in the Future Net after the tuning process.

A brief description of the hyperparameters listed in the table:

- Batch size: the amount of single entries for each batch.
- Dropout: the ratio of deactivated neurons.
- Leaky alpha: the value of the parameter that controls the negative slope in the leaky ReLU.
- L1 regularizer: the value of the parameter that controls L1 regularization.
- L2 regularizer: the value of the parameter that controls L2 regularization.
- Init distribution: the kind of distribution that generated the starting weights for the leaky ReLU or ReLU neurons, the He uniform distribution or the He normal distribution.

- Class weights: which weights were applied to class 0 and class 1.
- Resample type: which type of resample techniques were used, oversampling, undersampling, both of them or neither of them.
- Oversampling strategy: the proportion of the minority class over the majority class obtained after the resampling.

$$\frac{\text{Size of minority class}}{\text{Size of majority class}}$$

- K neighbors: a parameter that influences which data is picked to create the artificial data for the SMOTE algorithm.
- Optimizer: which kind of optimizer was implemented between SGD, SGD with momentum, SGD with Nesterov momentum, Adadelta or Adam.
- Initial learning rate: the starting learning rate.
- Decay steps: the number of steps required to trigger the reduction of the dynamic learning rate.
- Decay rate: the amount of the learning rate that is kept after each iteration of the reduction of the dynamic learning rate
- Last layer bias: if the last layer uses bias, or if it is set to 0.

The resample type indicates the type of resample that was applied, in this case, it was a Synthetic Minority Oversampling Technique oversampling with an oversampling strategy of 0.9. This means that the SMOTE created enough synthetic data to almost fully correct the class imbalance.

An interesting observation is that Leaky alpha = 0.0, this means that the network does not use the Leaky ReLU because, during the tuning process, it chose to switch to what is effectively a simple ReLU.

The Future Net was trained in 26 epochs while having a patience of 15 epochs.

3.3.3 Performance

The process of using the F1 score of the minority class of the validation dataset to evaluate the training of the Future Net allowed us to monitor in real time the improvement of the network and the preferred value that was used as the threshold.

Threshold development		
Epoch	F1 score	Threshold
0	0.734917	0.92
1	0.734670	0.90
5	0.734925	0.92
10	0.734946	0.92
15	0.734775	0.92
20	0.734820	0.90
25	0.734898	0.92

Table 3.6: In this table is shown the F1 score achieved by the Future Net during training. For the sake of brevity, only one in five of the epochs is shown, although the software updated all the values for each epoch.

As we can see from Table 3.6 the Future Net converged very quickly, with very limited improvements in performance in the following epochs. This is a desired behavior because it means that the model is fit to solve the problem and easily understands the data. The overall low complexity of the dataset and the use of the Adam optimizer (which is known to hasten the convergence of its models) may also have influenced the result. The best performance was achieved in epoch 9 with a threshold of 0.92 and a F1 score= 0.734967, with only an improvement of $5 \cdot 10^{-5}$ from the first epoch. Due to the lack of upgrades for the next 15 epochs, the training was interrupted. Even the value of the best threshold itself is very stable with occasional oscillations of 0.02, the value that was the minimum difference between the inspected thresholds during the training process.

The following tables contain the performance of the Future Net computed, respectively, on the test set and on the full dataset after the completion of the training.

Test dataset				
Class	Precision	Recall	F1 score	Support
0	0.95	0.94	0.94	2160159
1	0.72	0.76	0.74	451156
Overall accuracy				
0.91				

Full dataset				
Class	Precision	Recall	F1 score	Support
0	0.95	0.94	0.94	14398562
1	0.71	0.76	0.74	3010214
Overall accuracy				
0.91				

Table 3.7: The performance of the Future Net for both the test dataset and the full dataset. Class 0 represents the predictions that remain wrong, while class 1 represents the predictions that are confirmed with the new ground truth.

The results are overall positive. The classifier can recognize class 0 very well while being quite accurate with class 1. The recall higher than the precision for class 1 is positive because we are less interested in misclassifying class 0 rather than class 1, considering that this is an exploratory tool to recognize accurately class 1 data, false positives are more acceptable than false negatives. Considering the support (the quantity of data for each class) the higher amount of false positives makes only a small dent in the classification performance for class 0.

The consistency of the performance across the two datasets means that the network has not overfitted and generalizes the data well, this property confirms the high quality of the model, confirming that the model did not get stuck early in a local minima. This property is corroborated by the similar optimal threshold computed for both the datasets:

Best threshold	
Dataset	Threshold
Test dataset	0.940
Full dataset	0.935

Table 3.8: The best-performing threshold for both the test dataset and the full dataset.

Due to the similar threshold value, we can say that the behavior of the network is consistent and that it can generalize well, making the Future Net a reliable tool.

Comparison with the alternative

To verify the efficacy of the Future Net we must test it against the total score, to see if the network is more reliable for the identification of predictions that have a high probability of being corroborated in the future. Before using the total score to classify the prediction we will scale it using standard scaling on the dataset, all at once, in the same way that we used to prepare the total score for the Future Net. Then, the modulus of the minimum value is added to the total score, to make it fully positive, then is clipped to 2.5 and divided by 2.5 to bound it between 0 and 1. After these operations, we applied the same thresholding process that we used during the test of the Future Net, iterating between all possible thresholds between 0 and 1 but with a finer step of 0.0025, to be sure not to miss the best threshold and to solidify our test.

We can see the performance of both classification methods in the following table.

Classification prediction							
Test Dataset							
Total score				Future Net			
Class	Precision	Recall	F1 score	Class	Precision	Recall	F1 score
0	0.94	0.94	0.94	0	0.95	0.94	0.94
1	0.70	0.72	0.71	1	0.72	0.76	0.74
Accuracy		Threshold		Accuracy		Threshold	
0.90		0.0575		0.91		0.94	
Full dataset							
Total score				Future Net			
Class	Precision	Recall	F1 score	Class	Precision	Recall	F1 score
0	0.94	0.94	0.94	0	0.95	0.94	0.94
1	0.70	0.72	0.71	1	0.71	0.76	0.74
Accuracy		Threshold		Accuracy		Threshold	
0.90		0.0575		0.91		0.935	

Table 3.9: A comprehensive description of the performance of both the Future Net and the normalized total score used as classifier.

To further evaluate the performance of the two classifiers we draw the Receiver Operating Characteristic (ROC) [82] curve. The ROC curve plots the performance of a classifier through a graphical representation of the relationship between true positive rate (TPR) and false positive rate (FPR). The true positive rate (or recall) represents the capability of a classifier to recognize correctly the positive values, it is computed as follows:

$$TPR = \frac{\text{true positives}}{\text{true positives} + \text{false negatives}}$$

The false positive rate (or $1 -$ specificity) represents the capability of a classifier to recognize correctly the negative values, it is computed as follows:

$$FPR = \frac{\text{false positives}}{\text{false positives} + \text{true negatives}}$$

Due to the nature of these metrics, the optimal performance is found on the top left corner of the graph, because a classifier with that performance would achieve a $TPR = 1$ and a $FPR = 0$. This means that the classifier that is closer to the top left corner has the best performance of the other classifier, at its optimal threshold. For comparison, the random classifier, a classifier that represents a classification via random chance is also represented.

As we can see in Figure 3.8 the Future Net has the best performance at the optimal threshold, because the Future Net is the closest to the top left corner, confirming the previous data.

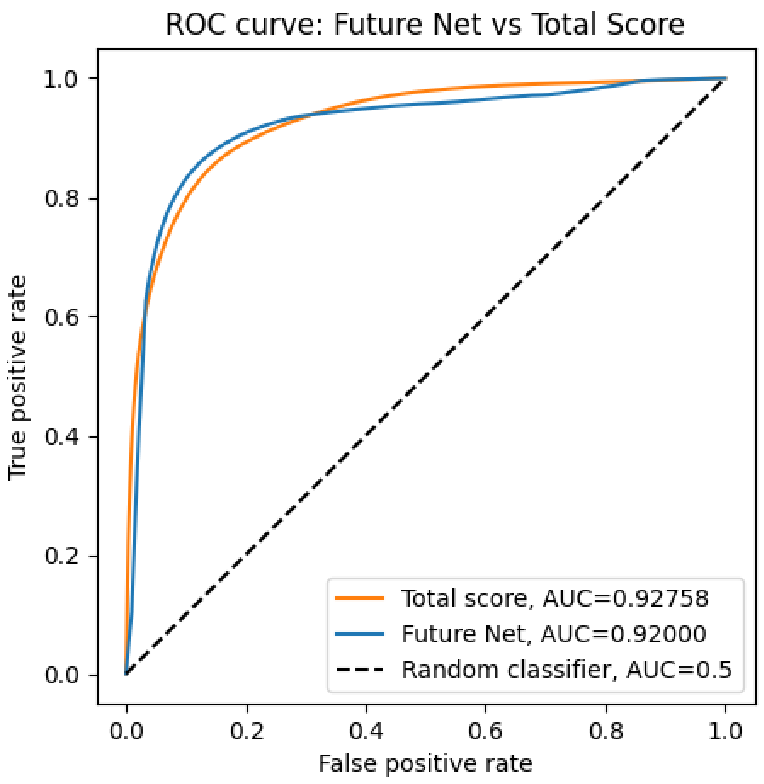


Figure 3.8: The ROC curve that compares the classification capabilities of the total score and the Future Net. This graph is computed using the classification of the entire dataset.

Another metric of interest in the ROC curve is the area under the curve (AUC) [82] of each classifier. The AUC is computed as follows:

$$AUC = \int_0^1 ROC(u)du$$

This metric tells us the overall reliability of the classifier. If the AUC of an investigated classifier is over the AUC of the random classifier it can be considered an improvement. The AUC is not strictly connected to the performance of the classifier because it evaluates the aggregated evaluation performance along the curve and does not focus on the threshold of the best performance. This means that it should not be used to directly decide which of the two classifiers is the best, and that even if the AUC of the total score (0.92578) has a slight edge over the AUC of the Future Net (0.92000) it should not take precedence on the tangible difference in performance.

The fact that both the total score used as a classifier and the Future Net have a high AUC corroborates the reliability of both classification methods, but the Future Net is the best-performing one and should be the classifier of choice.

Chapter 4

Conclusions

Overall, both the Main Net and the Future Net networks can be considered a success. The Main Net has a better performance in predicting the reliability of the Argot predictions than the spline. Due to this consistent upgrade, the Main Net will be used as the preferred method to select Argot predictions, improving significantly the output that the end-user receives.

The Future Net has a high enough performance to be used as an exploratory tool to search for predictions bearing a high probability to be anticipating the ground truth that may lead to impactful and efficient experimental research. Additional research on the topic may unveil predictive characteristics in the AFPs that anticipate the experimental evidence. Both of these networks will be used in the next version of Argot to provide more information to the software and further improve its performance.

Future perspectives

The research made to create the two neural networks was not exhaustive, better-performing architectures or a more rigorous tuning process may exist. Another option would be the use of a different machine learning technique altogether, although it is doubtful if a radically different approach would lead to better results for this use case.

Taking into account the constraints imposed in time and resources we can consider the project as a success.

Appendix

The representation of Model 11

On the next page, the architecture of Model 11 is depicted. It is the architecture used for both the the Main Net and the Future Net. The light orange parallelograms are standard blocks while the darker parallelograms are residual connection blocks (as shown in Figure 2.4). The last one is light blue because is a unique block, made only by a single dense layer. A small label under each individual block represents the width of that specific dense layer. The component of the residual connection block that performs the sum is drawn separately to clarify the connections.

Three blocks (two standard blocks and one residual connection block, in this order) form a group of blocks. Each group of blocks is named from D1 to D17, but D17 is the only one with a different structure. From each group of blocks starts a residual connection that connects to other parts of the network. Each connection is color-coded (except for the ones that lead to the same block, which are always grey), and the arrows indicate the flow of information. Each connection starts right after the first dense layer of each group to connect to other sum nodes.

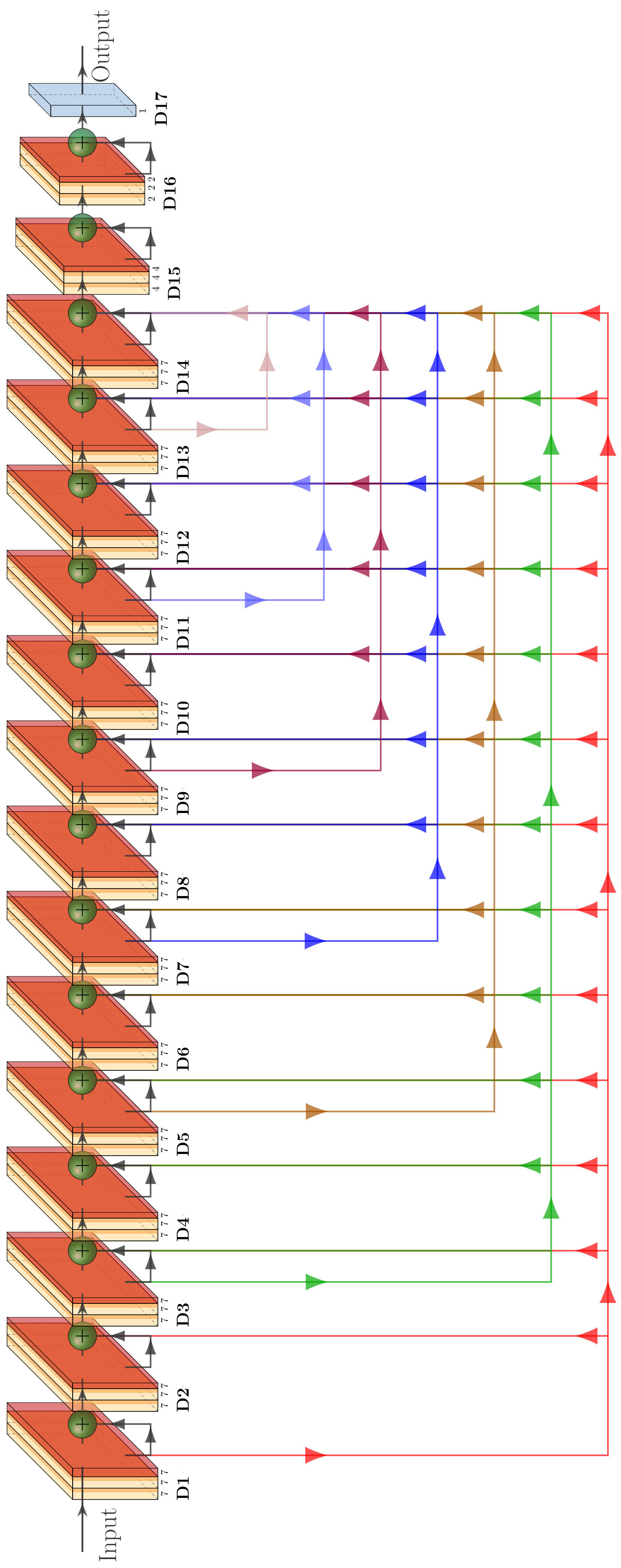


Figure 4.1: A representation of the architecture of Model 11.

Bibliography

[1] A. Goffeau, B. G. Barrell, H. Bussey, *et al.*, «Life with 6000 genes», *Science*, vol. 274, no. 5287, pp. 546–567, 1996. DOI: 10.1126/science.274.5287.546. eprint: https://www.science.org/doi/pdf/10.1126/science.274.5287.546. [Online]. Available: https://www.science.org/doi/abs/10.1126/science.274.5287.546.

[2] R. K. Wilson, «How the worm was won: The <em>c. elegans</em> genome sequencing project», *Trends in Genetics*, vol. 15, no. 2, pp. 51–58, Feb. 1999, ISSN: 0168-9525. DOI: 10.1016/S0168-9525(98)01666-7. [Online]. Available: https://doi.org/10.1016/S0168-9525(98)01666-7.

[3] F. Moraes and A. Góes, «A decade of human genome project conclusion: Scientific diffusion about our genome knowledge», *Biochemistry and molecular biology education*, vol. 44, 2016. DOI: https://doi.org/10.1002/bmb.20952.

[4] S. Schulze-Kremer, «Ontologies for molecular biology», *Pac Symp Biocomput.*, pp. 695–706, 1998. [Online]. Available: https://pubmed.ncbi.nlm.nih.gov/9697223/.

[5] G. O. Consortium, «The Gene Ontology (GO) database and informatics resource», *Nucleic Acids Research*, vol. 32, no. suppl$_1$, pp. D258–D261, Jan. 2004, ISSN: 0305-1048. DOI: 10.1093/nar/gkh036. eprint: https://academic.oup.com/nar/article-pdf/32/suppl_1/D258/7621365/gkh036.pdf. [Online]. Available: https://doi.org/10.1093/nar/gkh036.

[6] *Gene Ontology online documentation*, 2000. [Online]. Available: https://geneontology.org/docs/ontology-documentation/.

[7] N. Zhou, Y. Jiang, and T. R. B. et al., «The CAFA challenge reports improved protein function prediction and new functional annotations for hundreds of genes through experimental screens», *Genome Biology*, vol. 20, 2019. DOI: https://doi.org/10.1186/s13059-019-1835-8.

[8] E. Ispano, F. Bianca, E. Lavezzo, and S. Toppo, «An Overview of Protein Function Prediction Methods: A Deep Learning Perspective», *Current Bioinformatics*, vol. 18, no. 8, 2023. DOI: 10.2174/1574893618666230505103556.

[9] T. U. Consortium, «UniProt: the Universal Protein Knowledgebase in 2023», *Nucleic Acids Research*, vol. 51, no. D1, pp. D523–D531, Nov. 2022, ISSN: 0305-1048. DOI: 10.1093/nar/gkac1052. eprint: https://academic.oup.com/nar/article-pdf/51/D1/D523/48441158/gkac1052.pdf. [Online]. Available: https://doi.org/10.1093/nar/gkac1052.

[10] D. Cozzetto, F. Minneci, H. Currant, and D. T. Jones, «Ffpred 3: Feature-based function prediction for all gene ontology domains», *Scientific reports*, vol. 6, no. 1, p. 31 865, 2016.

[11] M. Kulmanov and R. Hoehndorf, «Deepgoplus: Improved protein function prediction from sequence», *Bioinformatics*, vol. 36, no. 2, pp. 422–429, 2020.

[12] E. Lavezzo, M. Falda, P. Fontana, L. Bianco, and S. Toppo, «Enhancing protein function prediction with taxonomic constraints – the argot2.5 web server», *Methods*, vol. 93, pp. 15–23, 2016, Computational protein function predictions, ISSN: 1046-2023. DOI: `https://doi.org/10.1016/j.ymeth.2015.08.021`. [Online]. Available: `https://www.sciencedirect.com/science/article/pii/S1046202315300657`.

[13] M. Falda, A. P. Stefano Toppo, B. D. C. Enrico Lavezzo, E. C. Andrea Facchinetti, and P. F. Riccardo Velasco, «Argot2: a large scale function prediction tool relying on semantic similarity of weighted Gene Ontology terms», *BMC Bioinformatics*, vol. 13(Suppl4), no. 13, 2011. DOI: `doi.org/10.1186/1471-2105-13-S4-S14`.

[14] W. Li and A. Godzik, «Cd-hit: a fast program for clustering and comparing large sets of protein or nucleotide sequences», *Bioinformatics*, vol. 22, no. 13, pp. 1658–1659, May 2006, ISSN: 1367-4803. DOI: `10.1093/bioinformatics/btl158`. eprint: `https://academic.oup.com/bioinformatics/article-pdf/22/13/1658/48838763/bioinformatics\_22\_13\_1658.pdf`. [Online]. Available: `doi.org/10.1093/bioinformatics/btl158`.

[15] S. F. Altschul, T. L. Madden, A. A. Schäffer, *et al.*, «Gapped blast and psi-blast: A new generation of protein database search programs», *Nucleic Acids Research*, vol. 25, no. 17, pp. 3389–3402, 1997, Cited by: 61939; All Open Access, Bronze Open Access. DOI: `10.1093/nar/25.17.3389`. [Online]. Available: `https://www.scopus.com/inward/record.uri?eid=2-s2.0-0030801002&doi=10.1093%2fnar%2f25.17.3389&partnerID=40&md5=ca92b4a3c6c209abea8b2c59b4b887e3`.

[16] S. R. Eddy, «Accelerated profile hmm searches», *PLOS Computational Biology*, vol. 7, no. 10, pp. 1–16, Oct. 2011. DOI: `10.1371/journal.pcbi.1002195`. [Online]. Available: `https://doi.org/10.1371/journal.pcbi.1002195`.

[17] R. D. Finn, A. Bateman, J. Clements, *et al.*, «Pfam: the protein families database», *Nucleic Acids Research*, vol. 42, no. D1, pp. D222–D230, Nov. 2013, ISSN: 0305-1048. DOI: `10.1093/nar/gkt1223`. eprint: `https://academic.oup.com/nar/article-pdf/42/D1/D222/3643441/gkt1223.pdf`. [Online]. Available: `https://doi.org/10.1093/nar/gkt1223`.

[18] D. (Clark), D. J.I., and C. E.C. Mungall, «Formalization of taxon-based constraints to detect inconsistencies in annotation and ontology development», *BMC Bioinformatics*, vol. 6, Apr. 2010. DOI: `doi.org/10.1186/1471-2105-11-530`.

[19] M. Falda, E. Lavezzo, P. Fontana, *et al.*, «Eliciting the functional taxonomy from protein annotations and taxa», *Scientific Reports*, vol. 6, Aug. 2016. DOI: `10.1038/srep31971`.

[20] P. Resnik, «Semantic similarity in a taxonomy: An information-based measure and its application to problems of ambiguity in natural language», *Journal of Artificial Intelligence Research*, vol. 11, 1999. DOI: `doi.org/10.1613/jair.514`.

[21] Y. Jiang, R. Oron, T. Clark, *et al.*, «An expanded evaluation of protein function prediction methods shows an improvement in accuracy», Sep. 2016. DOI: `10.17863/CAM.4487`.

[22] S. Shanmuganathan, «Artificial neural network modelling: An introduction», in *Artificial Neural Network Modelling*, S. Shanmuganathan and S. Samarasinghe, Eds. Cham: Springer International Publishing, 2016, pp. 1–14, ISBN: 978-3-319-28495-8. DOI: `10.1007/978-3-319-28495-8_1`. [Online]. Available: `https://doi.org/10.1007/978-3-319-28495-8_1`.

[23] D. Kriesel, *A Brief Introduction to Neural Networks*. 2007. [Online]. Available: `available%20at%20http://www.dkriesel.com`.

[24] W. Cannon, «Biographical memoir: Henry pickering bowdich», *National Academy of Sciences*, vol. 17, pp. 181–196, 1922.

[25] J. R. Hughes, «Post-tetanic potentiation», *Physiological reviews*, vol. 38, no. 1, pp. 91–113, 1958.

[26] J. Schmidhuber, «Deep learning in neural networks: An overview», *Neural Networks*, vol. 61, pp. 85–117, 2015, ISSN: 0893-6080. DOI: `https://doi.org/10.1016/j.neunet.2014.09.003`. [Online]. Available: `https://www.sciencedirect.com/science/article/pii/S0893608014002135`.

[27] M. Alloghani, D. Al-Jumeily, J. Mustafina, A. Hussain, and A. J. Aljaaf, «A systematic review on supervised and unsupervised machine learning algorithms for data science», in *Supervised and Unsupervised Learning for Data Science*, M. W. Berry, A. Mohamed, and B. W. Yap, Eds. Cham: Springer International Publishing, 2020, pp. 3–21, ISBN: 978-3-030-22475-2. DOI: `10.1007/978-3-030-22475-2_1`. [Online]. Available: `https://doi.org/10.1007/978-3-030-22475-2_1`.

[28] R. S. Sutton and A. G. Barto, *Reinforcement learning, second edition: an introduction*. 2018, ISBN: 0262352702, 9780262352703.

[29] N. Ketkar and J. Moolayil, «Feed-forward neural networks», in *Deep Learning with Python: Learn Best Practices of Deep Learning Models with PyTorch*. Berkeley, CA: Apress, 2021, pp. 93–131, ISBN: 978-1-4842-5364-9. DOI: `10.1007/978-1-4842-5364-9_3`. [Online]. Available: `https://doi.org/10.1007/978-1-4842-5364-9_3`.

[30] K. He, X. Zhang, S. Ren, and J. Sun, «Deep residual learning for image recognition», in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2016.

[31] W. Wang, Y. Huang, Y. Wang, and L. Wang, «Generalized autoencoder: A neural network framework for dimensionality reduction», in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, Jun. 2014.

[32] D. Bank, N. Koenigstein, and R. Giryes, «Autoencoders», in *Machine Learning for Data Science Handbook: Data Mining and Knowledge Discovery Handbook*, L. Rokach, O. Maimon, and E. Shmueli, Eds. Cham: Springer International Publishing, 2023, pp. 353–374, ISBN: 978-3-031-24628-9. DOI: `10.1007/978-3-031-24628-9_16`. [Online]. Available: `https://doi.org/10.1007/978-3-031-24628-9_16`.

[33] K. Venkatesh, K. Mohanasundaram, and V. Pothyachi, «8 - regression tasks for machine learning», in *Statistical Modeling in Machine Learning*, T. Goswami and G. Sinha, Eds., Academic Press, 2023, pp. 133–157, ISBN: 978-0-323-91776-6. DOI: https://doi.org/10.1016/B978-0-323-91776-6.00009-9. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780323917766000099.

[34] D. Bank, N. Koenigstein, and R. Giryes, *Machine Learning for Data Science Handbook: Data Mining and Knowledge Discovery Handbook*, L. Rokach, O. Maimon, and E. Shmueli, Eds. 2023, ISBN: 978-3-031-24628-9.

[35] K. Zhang, W. Zuo, Y. Chen, D. Meng, and L. Zhang, «Beyond a gaussian denoiser: Residual learning of deep cnn for image denoising», *IEEE Transactions on Image Processing*, vol. 26, no. 7, pp. 3142–3155, 2017. DOI: 10.1109/TIP.2017.2662206.

[36] S. Nah, T. H. Kim, and K. M. Lee, «Deep multi-scale convolutional neural network for dynamic scene deblurring», in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 257–265. DOI: 10.1109/CVPR.2017.35.

[37] Y. Ho and S. Wookey, «The real-world-weight cross-entropy loss function: Modeling the costs of mislabeling», *IEEE Access*, vol. 8, pp. 4806–4813, 2020. DOI: 10.1109/ACCESS.2019.2962617.

[38] R. Rojas, «The backpropagation algorithm», in *Neural Networks: A Systematic Introduction*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, pp. 149–182, ISBN: 978-3-642-61068-4. DOI: 10.1007/978-3-642-61068-4_7. [Online]. Available: https://doi.org/10.1007/978-3-642-61068-4_7.

[39] M. A. Nielsen, *Neural Networks and Deep Learning*. Determination Press, 2015.

[40] L. Bottou, «Large-scale machine learning with stochastic gradient descent», in *Proceedings of COMPSTAT'2010*, Y. Lechevallier and G. Saporta, Eds., Heidelberg: Physica-Verlag HD, 2010, pp. 177–186, ISBN: 978-3-7908-2604-3.

[41] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[42] Y. Liu, Y. Gao, and W. Yin, *An improved analysis of stochastic gradient descent with momentum*, 2020. arXiv: 2007.07989 [math.OC].

[43] T. Tieleman and G. Hinton, «Lecture 6.5-rmsprop, coursera: Neural networks for machine learning», *University of Toronto, Technical Report*, vol. 6, 2012.

[44] J. Duchi, E. Hazan, and Y. Singer, «Adaptive subgradient methods for online learning and stochastic optimization.», *Journal of machine learning research*, vol. 12, no. 7, 2011.

[45] D. P. Kingma and J. Ba, «Adam: A method for stochastic optimization», *arXiv preprint arXiv:1412.6980*, 2014.

[46] L. N. Smith, «A disciplined approach to neural network hyper-parameters: Part 1 - learning rate, batch size, momentum, and weight decay», *CoRR*, vol. abs/1803.09820, 2018. arXiv: 1803.09820. [Online]. Available: http://arxiv.org/abs/1803.09820.

[47] M. Stone, «Cross-validatory choice and assessment of statistical predictions», *Journal of the royal statistical society series b-methodological*, vol. 36, pp. 111–133, 1976. [Online]. Available: `https://api.semanticscholar.org/CorpusID:62698647`.

[48] P. Ramachandran, B. Zoph, and Q. V. Le, *Searching for activation functions*, 2017. arXiv: `1710.05941 [cs.NE]`.

[49] A. K. Dubey and V. Jain, «Comparative study of convolution neural network's relu and leaky-relu activation functions», in *Applications of Computing, Automation and Wireless Systems in Electrical Engineering*, S. Mishra, Y. R. Sood, and A. Tomar, Eds., Singapore: Springer Singapore, 2019, pp. 873–880, ISBN: 978-981-13-6772-4.

[50] S. R. Dubey, S. K. Singh, and B. B. Chaudhuri, «Activation functions in deep learning: A comprehensive survey and benchmark», *Neurocomputing*, vol. 503, pp. 92–108, 2022, ISSN: 0925-2312. DOI: `https://doi.org/10.1016/j.neucom.2022.06.111`. [Online]. Available: `https://www.sciencedirect.com/science/article/pii/S0925231222008426`.

[51] A. Apicella, F. Donnarumma, F. Isgrò, and R. Prevete, «A survey on modern trainable activation functions», *Neural Networks*, vol. 138, pp. 14–32, 2021, ISSN: 0893-6080. DOI: `https://doi.org/10.1016/j.neunet.2021.01.026`. [Online]. Available: `https://www.sciencedirect.com/science/article/pii/S0893608021000344`.

[52] A. D. Rasamoelina, F. Adjailia, and P. Sinčák, «A review of activation function for artificial neural network», in *2020 IEEE 18th World Symposium on Applied Machine Intelligence and Informatics (SAMI)*, 2020, pp. 281–286. DOI: `10.1109/SAMI48414.2020.9108717`.

[53] J. Xu, Z. Li, B. Du, M. Zhang, and J. Liu, «Reluplex made more practical: Leaky relu», in *2020 IEEE Symposium on Computers and Communications (ISCC)*, 2020, pp. 1–7. DOI: `10.1109/ISCC50000.2020.9219587`.

[54] X. Qi, Y. Mei, X. Mei, R. Chellali, and S. Yang, «Comparative analysis of the linear regions in relu and leakyrelu networks», in *Neural Information Processing*, vol. 8, 2024, pp. 528–539. DOI: `https://doi.org/10.1007/978-981-99-8132-8_40`.

[55] F. He, T. Liu, and D. Tao, «Why resnet works? residuals generalize», *IEEE Transactions on Neural Networks and Learning Systems*, vol. 31, no. 12, pp. 5349–5362, 2020. DOI: `10.1109/TNNLS.2020.2966319`.

[56] S. Ioffe and C. Szegedy, «Batch normalization: Accelerating deep network training by reducing internal covariate shift», *CoRR*, vol. abs/1502.03167, 2015. arXiv: `1502.03167`. [Online]. Available: `http://arxiv.org/abs/1502.03167`.

[57] D. Vidaurre, C. Bielza, and P. Larranaga, «A survey of l1 regression», *International Statistical Review*, vol. 81, no. 3, pp. 361–387, 2013.

[58] A. Gosain and S. Sardana, «Handling class imbalance problem using oversampling techniques: A review», in *2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, 2017, pp. 79–85. DOI: `10.1109/ICACCI.2017.8125820`.

[59] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, «Smote: Synthetic minority over-sampling technique», *J. Artif. Int. Res.*, vol. 16, no. 1, pp. 321–357, Jun. 2002, ISSN: 1076-9757.

[60] C. Dessimoz, N. Škunca, and P. D. Thomas, «Cafa and the open world of protein function predictions», *Trends in Genetics*, vol. 29, no. 11, pp. 609–610, 2013.

[61] P. Wasserman and T. Schwartz, «Neural networks. ii. what are they and why is everybody so interested in them now?», *IEEE Expert*, vol. 3, no. 1, pp. 10–15, 1988. DOI: `10.1109/64.2091`.

[62] C. Sun, A. Shrivastava, S. Singh, and A. Gupta, «Revisiting unreasonable effectiveness of data in deep learning era», in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, Oct. 2017.

[63] S. Sukhbaatar and R. Fergus, «Learning from noisy labels with deep neural networks», *arXiv preprint arXiv:1406.2080*, vol. 2, no. 3, p. 4, 2014.

[64] J. Benitez, J. Castro, and I. Requena, «Are artificial neural networks black boxes?», *IEEE Transactions on Neural Networks*, vol. 8, no. 5, pp. 1156–1164, 1997. DOI: `10.1109/72.623216`.

[65] S. Wang, W. Liu, J. Wu, L. Cao, Q. Meng, and P. J. Kennedy, «Training deep neural networks on imbalanced data sets», in *2016 International Joint Conference on Neural Networks (IJCNN)*, 2016, pp. 4368–4374. DOI: `10.1109/IJCNN.2016.7727770`.

[66] C. F. G. D. Santos and J. P. Papa, «Avoiding overfitting: A survey on regularization methods for convolutional neural networks», *ACM Comput. Surv.*, vol. 54, no. 10s, Sep. 2022, ISSN: 0360-0300. DOI: `10.1145/3510413`. [Online]. Available: `https://doi.org/10.1145/3510413`.

[67] G. Consortium, *Gene ontology release statics from 06/2024*, Jun. 2024. [Online]. Available: `https://geneontology.org/stats.html`.

[68] G. Van Rossum and F. L. Drake, *Python 3 Reference Manual.* Scotts Valley, CA: CreateSpace, 2009, ISBN: 1441412697.

[69] F. Chollet *et al.*, *Keras*, `https://keras.io`, 2015.

[70] Martín Abadi, Ashish Agarwal, Paul Barham, *et al.*, *TensorFlow: Large-scale machine learning on heterogeneous systems*, Software available from tensorflow.org, 2015. [Online]. Available: `https://www.tensorflow.org/`.

[71] J. D. Hunter, «Matplotlib: A 2d graphics environment», *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007. DOI: `10.1109/MCSE.2007.55`.

[72] T. pandas development team, *Pandas-dev/pandas: Pandas*, version latest, Dec. 2023. DOI: `10.5281/zenodo.10304236`. [Online]. Available: `https://doi.org/10.5281/zenodo.10304236`.

[73] C. R. Harris, K. J. Millman, S. J. van der Walt, *et al.*, «Array programming with NumPy», *Nature*, vol. 585, no. 7825, pp. 357–362, Sep. 2020. DOI: `10.1038/s41586-020-2649-2`. [Online]. Available: `https://doi.org/10.1038/s41586-020-2649-2`.

[74] F. Pedregosa, G. Varoquaux, A. Gramfort, *et al.*, «Scikit-learn: Machine learning in Python», *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[75] K. He, X. Zhang, S. Ren, and J. Sun, «Delving deep into rectifiers: Surpassing human-level performance on imagenet classification», *CoRR*, vol. abs/1502.01852, 2015. arXiv: `1502.01852`. [Online]. Available: `http://arxiv.org/abs/1502.01852`.

[76] S. K. Kumar, «On weight initialization in deep neural networks», *CoRR*, vol. abs/1704.08863, 2017. arXiv: `1704.08863`. [Online]. Available: `http://arxiv.org/abs/1704.08863`.

[77] ducha-aiki, *Experimental batch normalization evaluation on image-net 2012*, 2016. [Online]. Available: `https://github.com/ducha-aiki/caffenet-benchmark/blob/master/batchnorm.md`.

[78] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar, «Hyperband: A novel bandit-based approach to hyperparameter optimization», *Journal of Machine Learning Research*, vol. 18, no. 185, pp. 1–52, 2018. [Online]. Available: `http://jmlr.org/papers/v18/16-558.html`.

[79] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollar, «Focal loss for dense object detection», in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, Oct. 2017.

[80] B. L. Welch, «The generalization of 'student's'problem when several different population varlances are involved», *Biometrika*, vol. 34, no. 1-2, pp. 28–35, 1947.

[81] G. D. Ruxton, «The unequal variance t-test is an underused alternative to Student's t-test and the Mann–Whitney U test», *Behavioral Ecology*, vol. 17, no. 4, pp. 688–690, May 2006, ISSN: 1045-2249. DOI: `10.1093/beheco/ark016`. eprint: `https://academic.oup.com/beheco/article-pdf/17/4/688/17275561/ark016.pdf`. [Online]. Available: `https://doi.org/10.1093/beheco/ark016`.

[82] L. Gonçalves, A. Subtil, M. R. Oliveira, and P. de Zea Bermudez, «Roc curve estimation: An overview», *REVSTAT-Statistical journal*, vol. 12, no. 1, pp. 1–20, 2014.