



UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA



Escola Tècnica
Superior d'Enginyeria
Informàtica

Using neural networks based on epigenomic maps for predicting the transcriptional regulation measured by CRISPR/Cas9.

Trabajo Fin de Grado en Ingeniería Informática - Escuela Técnica
Superior de Ingeniería Informática

July 2016

Alejandro Barberá Mourelle

Mentors:

J. Alberto Conejero

Diego Orzaez (Experimental mentor)

Abstract

Because of the great impact that the genomic editing with CRISPR/CAS9 has had in the recent years, and the great advances that it brings to biotechnology a great need of information has arisen. However researches struggle to find a definite pattern with these experiments making a very long process of trial and error to find an optimal solution for a particular experiment.

With this project we intend to optimize the genomic edition with the newest advance CRISPR/Cas9, to find the optimal insertion site we design a mathematical model based on neural networks. During this process we had to deal with huge amount of information from the genome so we had to develop a way to filter and handle it efficiently.

For this project we are going to focus in *Arabidopsis Thaliana* which is a very common plant in genomic edition and has many resources available online.

How I started

In June 2014, one of my mentors (Alberto Conejero), offered me to get involved with the UPV team to participate in the iGem which is a synthetic biology competition sponsored by the MIT. It was at that moment that I began to show interest in biology and how to use computer science skills to optimize some tasks. My contribution to the team was a simulation of the laboratory environment in Minecraft so that we could use it as a teaching tool to introduce people at younger ages to biochemistry.

At the end of the summer, after having presented our project in Boston, I contacted Diego Orzaez who had been the responsible for the iGem team, as he leads the CSIC laboratory working with plants in order to continue working with him on a TFG that combines computer science and biology.

Acknowledgements

For this project I will like to thank the following:

My mentors Alberto Conejero and Diego Orzaez

For their time and the faith in me they have shown.

My mom and dad

For dealing with my ranting about the problems faced in this project.

David Torrents

Computational genomics group manager at Life Sciences department in Barcelona Supercomputing Center. For hiring me and allowing me to continue working in a very interesting field.

Pilar Baldominos

For teaching me the basis of biology over and over again and keeping up with hours and hours of ranting about why people know very little about programming.

All dreams eventually come to an end when the dreamer wakes.

-Gilgamesh, Fate Zero.

He who knows nothing can understand nothing.

-Ansem, Kingdom Hearts.

Contents

| | | |
|----------|---|-----------|
| 1 | Introduction | 1 |
| 1.1 | Structure of introduction | 1 |
| 1.2 | Introduction to genomics | 1 |
| 1.2.1 | Genomes: basic knowledge | 1 |
| 1.2.1.1 | Physical Structure | 2 |
| 1.2.1.2 | Biological Structure | 3 |
| 1.2.2 | Evolution of genome editing | 4 |
| 1.2.3 | CRISPR/Cas9 system | 4 |
| 1.2.4 | Optimization parameters for CRISPR/Cas9 DNA binding | 6 |
| 1.3 | Handling big data sources | 7 |
| 1.4 | Multi-Layer Neural Networks | 9 |
| 1.5 | Objectives | 9 |
| 1.6 | Memory structure | 10 |
| 2 | Neural Networks | 11 |
| 2.1 | Introduction | 11 |
| 2.2 | Structure | 11 |
| 2.2.1 | Types of neural networks | 12 |
| 2.3 | Activation Functions | 13 |
| 2.4 | Foward Propagation | 14 |
| 2.5 | Backward Propagation (Training) | 15 |
| 2.5.1 | Stop conditions | 17 |
| 2.5.2 | Data samples | 18 |
| 3 | The Mathematical Model | 19 |
| 3.1 | Introduction | 19 |
| 3.1.1 | Selected data | 19 |
| 3.2 | The data | 19 |
| 3.2.1 | Some file formats | 19 |
| 3.2.1.1 | The BED format | 20 |
| 3.2.1.2 | The GFF3 format | 20 |
| 3.2.2 | Extracting the data | 21 |
| 3.2.2.1 | Genes | 21 |

| | | |
|----------|---|-----------|
| 3.2.2.2 | Occupancy | 22 |
| 3.2.3 | Selecting correct insertion positions | 22 |
| 3.2.4 | Normalizing the data | 22 |
| 3.2.5 | Using the neural network | 23 |
| 3.2.6 | Selecting the best values | 23 |
| 3.3 | Adapting results from humans | 23 |
| 3.4 | Results | 24 |
| 4 | CRISPR/Cas9 Neural Net | 25 |
| 4.1 | Introduction | 25 |
| 4.2 | Activation Function | 26 |
| 4.3 | Handling inputs | 27 |
| 4.3.1 | Final setup | 28 |
| 4.4 | Optimizing the structure | 28 |
| 4.4.1 | Introduction to genetic algorithms | 29 |
| 4.4.2 | Our genetic algorithm | 29 |
| 5 | Our Application | 33 |
| 5.1 | Roles | 33 |
| 5.1.1 | The researcher | 33 |
| 5.1.2 | The manager | 33 |
| 5.2 | Overview | 33 |
| 5.2.1 | Main window | 34 |
| 5.2.2 | Predict window | 34 |
| 5.2.3 | Train window | 36 |
| 6 | Looking Foward | 37 |
| 6.1 | The Neural Net | 37 |
| 6.2 | The Mathematical Model | 37 |
| 7 | References | 38 |

Chapter 1: Introduction

1.1 Structure of introduction

We will now explain how the introduction is going to be formatted. First we will do a short explanation on what a genome is, some methods on genome editing that were being used until now, the latest editing method with CRISPR/Cas9. Then we will continue to explain some method of software optimization that were made to handle the data and a brief introduction to neural networks. Finally I will state the objectives of this project aswell as the structure of the whole paper.

1.2 Introduction to genomics

The new developments in society and the improvements of life conditions have exponentially increased life expectancy. As a result, the global population is rapidly increasing and new challenges are arising. One of the most important issues to solve in a short time is to ensure the nourishment of the population. Genetic engineering offers a unique opportunity to increase production in a sustainable manner and also possibility the creation of varieties able to grow in hard environmental conditions. A new tool for this purpose has recently appeared and is revolutionizing the efficiency and easiness of plant genetic engineering, the CRISPR/Cas9 system. Thus, the optimization and a better understanding of this technique will provide an excellent tool for the future global challenges.

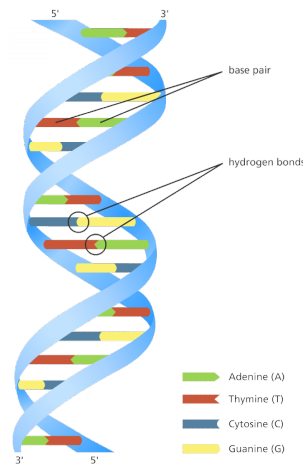
1.2.1 Genomes: basic knowledge

Biologic systems encode all the information needed for their correct development and maintenance in their genome formed by a sequence of four different molecules called nucleotides. Each nucleotide is composed of a sugar (deoxyribose), a phosphate and a nitrogenated base. There are four types of nitrogenated bases which determines the corresponding nucleotide: **adenine (A)**, **thymine (T)**, **cytosine (C)** and **guanine (G)**. The sequence will determine the function of each region and will be converted into biological effector molecules (proteins) following the genetic code which is common among all species.

1.2.1.1 Physical Structure

The DNA is composed of two strands of nucleotides joined together by the homology of the nucleotides in each strand. Adenine will match with thymine while cytosine will bind to guanine. In that way the possibilities of repair the correct sequence in case of an accident increase. The two strands are coiled around each other forming the well known structure of the double helix (Figure 0).

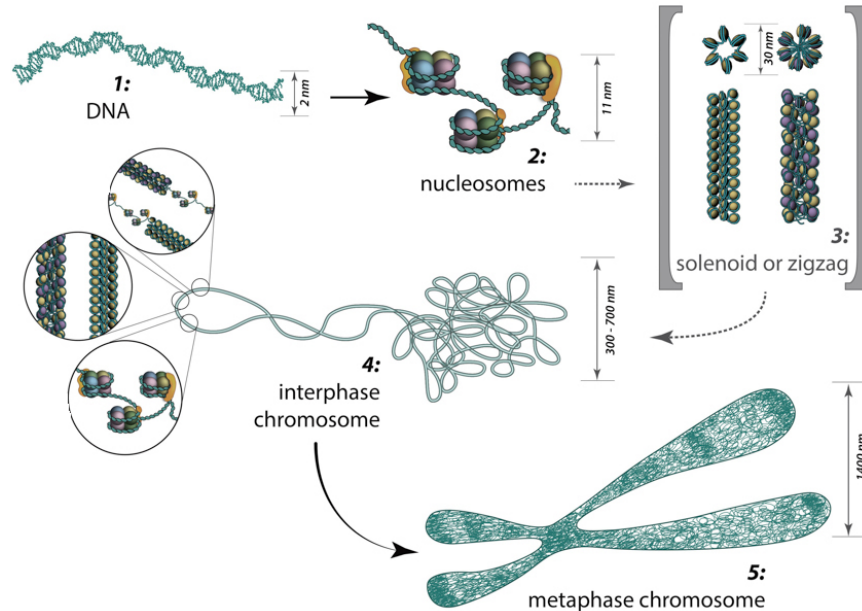
Figure 0: DNA structure



Source: <http://www.yourgenome.org/stories/unravelling-the-double-helix>

However this structure needs further folding in order to use the as less space as possible inside the cell. The double helix is wrapped by proteins which are positively charged in order to neutralize the charges of phosphates in a structure called nucleosome (Step 2 in Figure 1). The nucleosomes then ensamble in a zigzag or solenoid structure in order to compact more forming the chromatine fibers which are the structure of the chromosomes during normal cell development (Step 4 in Figure 1). In this state the are regions with higher or lower levels of compacting fibers according to the importance of the biological function of the region in the present time for the cell. This way the cell will have available the necessary genes depending on which action is going to do (growing, dividing, tissue differentiation ...). The highest compacting factor is while the cell is dividing as the chromosomes are being formed to ensure an even DNA division (Step 5 in Figure 1).

Figure 1: Steps in DNA Folding



Source: <https://www.mechanobio.info/topics/genome-regulation/dna-packaging/>

1.2.1.2 Biological Structure

The smallest form of DNA that still has a biological function is called “Gene”. Each gene has 3 regions:

Promoter

Marks the start of a gene, its sequence will determine whether it will express constitutively or under certain circumstances.

Coding region

It is the region of the gene that contains the instructions for the formation of a protein according to the laws of the genetic code where every 3 nucleotides will determine the linked amino acid.

Terminator

Marks the end of the gene.

1.2.2 Evolution of genome editing

Homologous recombination-dependent gene targeting

Homologous recombination is a natural process that occurs when there is damage in DNA. Consists of the union of 2 DNA fragments by the likeness of a fragment of their sequences. Genetic engineering has used this technique as a means of introducing an artificial sequence inside a genome. This technique is quite inefficient, also it can introduce the sequence in unwanted sites.

Recombinase-mediated site-specific gene integration

Recombinases are proteins common in prokaryotes and lower eukaryotes, in which they participate in various biological functions. They cut DNA into determined sequences. This was a way to try to improve efficiency of homologous recombination. However, they continued with the problem of inserting in unwanted sites.

Nuclease-mediated site-specific genome modification

In order to try and solve the issue with the position of integration chimeric proteins were created with a domain of an endogenous endonuclease able to produce double strand breaks and a domain of a protein able to recognize specific DNA sequences. Then the cut will be produced in the desired place and the homologous recombination will be enhance in order to solve the damage produced. This is the case of the zinc fingers and the TALEN's technology.

CRISPR/Cas9 system

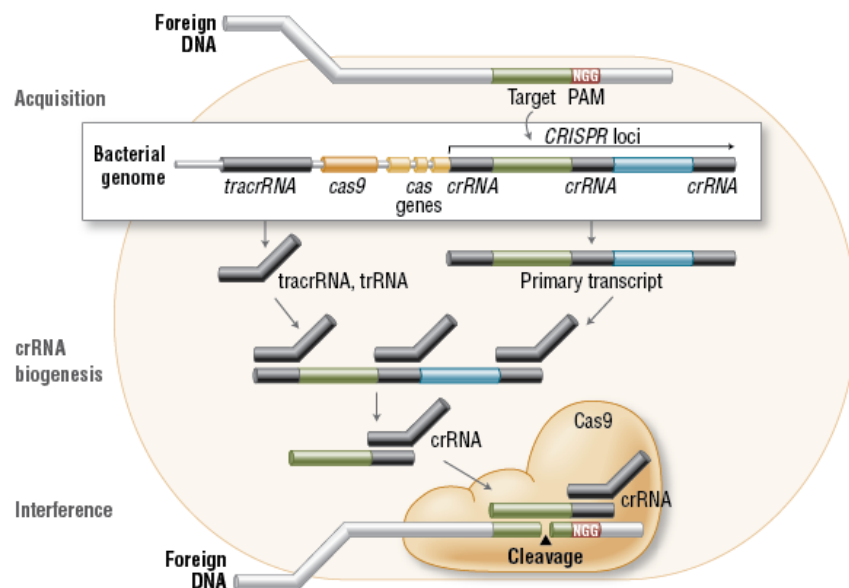
The latest tool for genomic edition. Based on the latest discovery of Francisco Mojica (for further information see *The Heroes of CRISPR*: [http://www.cell.com/cell/pdf/S0092-8674\(15\)01705-5.pdf](http://www.cell.com/cell/pdf/S0092-8674(15)01705-5.pdf)) that has recently been awarded with the "Jaime I award on basic research". This is what we will be using, and will be explained thoroughly in the next section.

1.2.3 CRISPR/Cas9 system

In order to better understand this tool is necessary to know its origin. CRISPR/Cas9 is an adaptive immunologic system present in most bacteria and archea as a defense against bacteriophages. The system is composed by a protein called Cas (CRISPR associated) endonuclease which is able to cleave the invader genome by recognition of a specific sequence given by the Clustered Regularly Interspaced Short Palindromic Repeats (CRISPR) loci (Marraffini, 2015). These loci is composed of short sequences (20 nucleotides) of exogenous

genomes that the bacteria has been acquiring during evolution, and allow it to recognize specifically the invading organism genome and cleave it. Flanking the cassette with this sequences there are short repetitive sequences (spacers). When this loci is transcribed, small CRISPR RNAs are produced by the cleavage of the cassettes and the spacers. They will form a dimeric structure by homologous recognition of spacers and each cassette. This interaction will allow the recognition of the Cas9 protein, which will be directed by the cassette sequence until the exogenous DNA in order to bind it, and subsequently cleave it (Figure 2) (H. Nishimasu 2014). The only additional requirement for efficient cleavage is the presence of a trinucleotide NGG in the exogenous DNA (Protospacer Adjacent Motif, PAM), immediately upstream of the 20 nucleotides target sequence. Based on this natural defensive system, a very potent biotechnological tool was developed. The spacer and the cassette were designed as a single RNA strand called the guided RNA (sgRNA) (M. Jinek2012). This very simple, easy-to-make molecule, the sgRNA, carries all the information required to “program” cas9 activity, directing it to any given site in the genome.

Figure 2: CRISPR/Cas9 working scheme.



Source: *CRISPR/Cas9 and Targeted Genome Editing: A New Era in Molecular Biology (New England Biolabs).*

This RNA-guided endonuclease provide the awaited specificity for directed cleavage of genomes in genetic engineering. Once the system has cut the genome, the host will recognize the DNA

damage and will induce the mechanisms to repair it. The most efficient one is the homologous recombination that is usually done with the other chromosome sequence. In the case of the genetic engineering the sequence that has to be inserted in the genome is introduced in the organism inside a plasmid or a sequence flanked by the homologous sequence of the sites where the CRISPR/Cas9 system has cleavage the genome. Hence the sequence is inserted in a directed and specific genome position. The main advantage of this directed edition, is the prevention of inserting the sequence in the middle of a coding region or a regulatory element that can unbalance the cell cycle in a malignant or not desired manner. This is the key element that makes CRISPR such a desired technique.

Cas9 can be also mutagenized to inactivate its cleavage activity, yet keeping its RNA-guided DNA binding activity (named as dead Cas9 or dCas9). This turns dCas9 into a programmable DNA-binding protein, which can be used to regulate gene expression at specific locations in the genome. Thus, it is not rare that in only four years from the first article using them as a biotechnological tool (M. Jinek2012), they have become the main trend in every aspect of actual biotechnology with 805 publications this year.

1.2.4 Optimization parameters for CRISPR/Cas9 DNA binding

The affordability and technical accessibility of this system for genome editing led to its wide adoption in research laboratories. Its widespread use allowed the identification of different efficiency levels between experiments with different targets. In order to solve this, many strategies have been proposed, as the reduction in one or two, the number of recognition nucleotides expecting that the effect on the stability of a single mismatch is increased (Y. Fu 2014). The efficiency is also affected not only by the sequence but also by the location and the conformation of the genome in the target region. The DNA in the nucleus is compacted by some protein complexes called nucleosomes. Depending on the position and conformation of this complexes the chromatin (the complex form by the DNA and all packaging proteins) can be more open or more compact. It is known that CRISPR system prefers open chromatin regions for its binding (Wu X 2014) (Kuscu C 2014). Efficiency is also increased when in the sgRNA at positions 17-20 there are purine bases instead of pirimidine ones (Doench JG, 2014).

With this, we theorise that using the **distance from the origin of the inserted gene** and the **nucleosome occupancy** we can develop a machine learning algorithm based on neural networks in order to be able to predict the genomic transcription after inserting a CRISPR.

1.3 Handling big data sources

For our objective we are going to extract data from independent sources. Each bit of information is extracted from a very big text file and then parsed into our custom made data structure, that we will explain later. This is a relative standard job, however due to the size of the files we have to design a very efficient way to search for those data without penalizing time but also taking into consideration that we are using simple desktop computers and not HPCs or clusters as we intend to use this project in a non-professional aswell as in a professional environment.

The first time we attempted to load the data in a sequential manner we run out of memory after a couple of hours of computing time, we are not talking about phisical memory but RAM. After some simple theoretical calculations, if we ignore the errors due to running out of memory, the estimated time of computation was over 50 hours, so we had to implement some ways of optimizing the code.

Some of the used techniques involve:

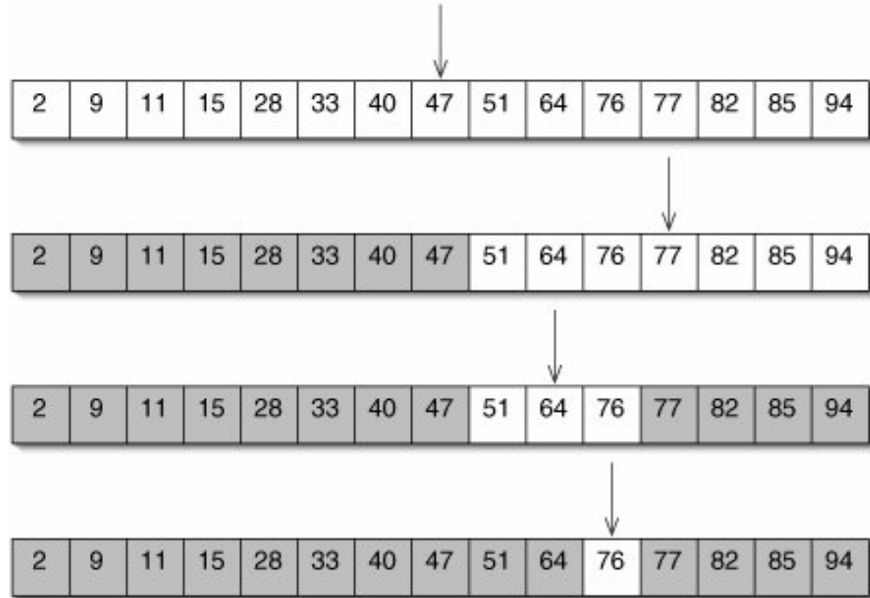
Parallelization

Because of the amount of individual instances that we are going to handle and the fact that they are indenpedant from each other we can divide the workload into segments and assign a thread to each to compute them in parallel. With this technique we could theoretically divide the computation time by the number of threads created.

Binary search

Some files are ordered in ascending order of position, which makes it really convinient to use a binary seach algorithm hence improving the time cost from linear $\theta(n)$ to logarithmic $O(\log(n))$.

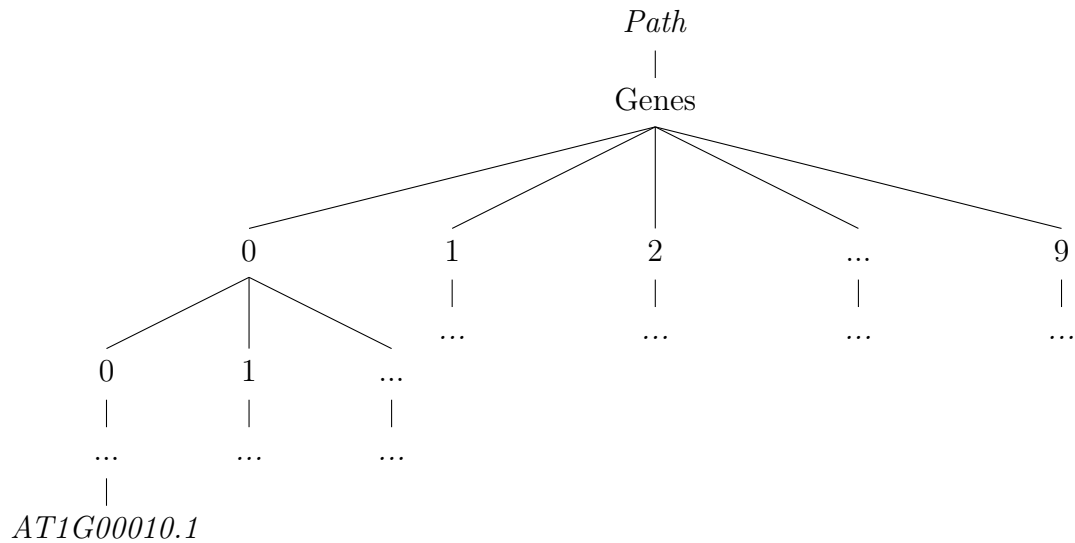
Figure 3: Visual representation of binary search for number 76.



Source: https://upload.wikimedia.org/wikipedia/commons/f/f7/Binary_search_into_array.png

Directory structure

In order to store these individual data structures instead of creating a single file with all the data in the same file, we created a set of directories that work as an index in order to access individual sets fast. The relative path to the gene we are looking for is the same as its name.



With this we go from a $\theta(n)$ search time for the file containing its data to $O(1)$.

Genetic Algorithms

This is a type of algorithm used for maximization used widely in artificial intelligence, it uses search heuristics to mimic *natural selection*. This will help us later on with optimizing the structure for our neural network.

After implementing the binary search and the parallelization of the loading we managed to retrieve all the data in under 3 minutes whereas if we did it sequentially a simple math calculation tells us that it could have taken over 5 days to execute.

1.4 Multi-Layer Neural Networks

In recent years machine learning has evolved greatly and learning algorithms have become popular, inside these algorithms we can find artificial neural networks (ANNs). They are inspired by biological neural networks, as we can find in the brain being responsible for the nervous systems in not living things, and are normally used as classifiers or predictor based on tagged training.

Artificial neural networks are made up of “neurons“ that are connected to other neurons by weighted directional paths. These connections have numeric weights that can be tuned based on experience, making neural nets able to adapt to changes in the “input“ data.

There are already some applications in biotechnology that use neural networks as classifiers and because of the lack of knowledge on what are the variables that affect this particular problem we think that the use of neural networks is optimal to create a prediction method for these experiments.

1.5 Objectives

Reduce the cost of finding the optimal position to target a CRISPR to achieve maximum/minimum expression of a given gene. For this we follow these steps:

1. Test if using the distance to the start of the inserted gene and the nucleosome occupancy around the point of insertion is enough to create a valid predictor using neural networks.
2. Create an interface to handle big amount of data with the available resources of the computer.

3. Create a neural network to find the optimal site to insert a CRISPR/Cas9.
4. Create an interface to interact with the neural net.

1.6 Memory structure

This memory is structured as follows. The second chapter explains all about neural networks and how to use and train them, the third part talks about the mathematical model we implemented, the fourth explains how our neural net is built, the fifth shows the use of our application and the sixth and final segment (because its too brief to call it a chapter) will pinpoint some ways we could improve this application in the future and how to adapt it in case a different approach is needed.

Chapter 2: Neural Networks

2.1 Introduction

In machine learning and cognitive science, artificial neural networks (ANNs) belong in models inspired by biological neural networks (the central nervous systems of animals, in particular the brain) which are used to estimate or approximate functions that can depend on a large number of inputs and are generally unknown. Artificial neural networks are generally presented as systems of interconnected "neurons" which exchange messages between each other. The connections have numeric weights that can be tuned based on experience, making neural nets adaptive to inputs and capable of learning.

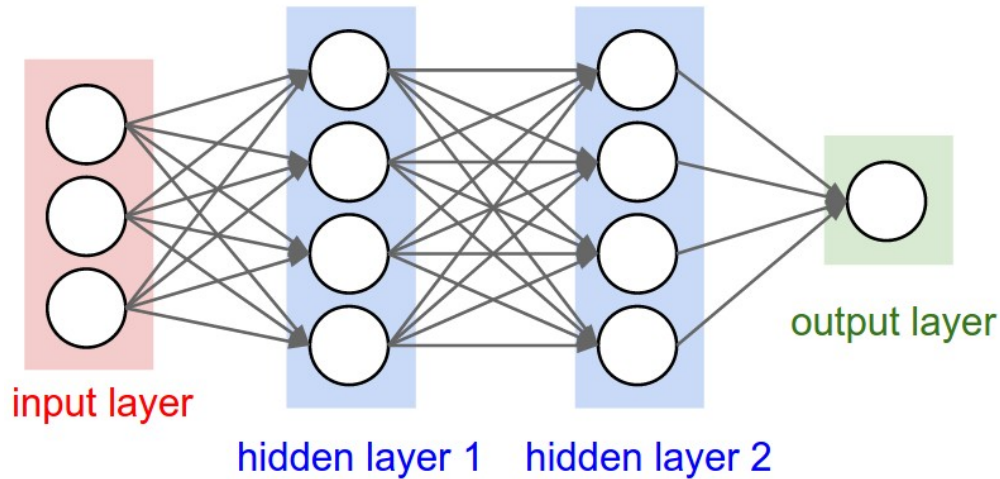
For example, a neural network for handwriting recognition is defined by a set of input neurons which may be activated by the pixels of an input image. After being weighted and transformed by a function (determined by the network's designer), the activations of these neurons are then passed on to other neurons. This process is repeated until the output neuron that determines which character was read is activated.

2.2 Structure

A standard neural network is made out of nodes (or neurons) interconnected with each other, they are grouped in ordered layers of one or more nodes. Each layer is completely connected to the next layer.

Each neural network contains an input layer, an output layer and 0 or more hidden layers with 1 or more neurons in each layer. The number of hidden layers and the number of nodes in them can determine how the network will behave.

A very simple neural network.



Source: <http://cs231n.github.io/convolutional-networks/>

2.2.1 Types of neural networks

Based on how the nodes of a network are linked with each other we can differentiate a few types of neural networks.

Normal

This is not actually an established type of neural network because it is the most basic structure of them all. Here the nodes from a layer are fully connected only to the nodes in the next layer of the network. This will be the structure we will be using in our project.

Recursive

A recurrent neural network is a kind of deep neural network created by applying the same set of weights recursively over a structure creating loops, to produce a structured prediction over variable-length input, or a scalar prediction on it, by traversing a given structure in topological order.

Fully recurrent

In this setup all nodes are connected fully with each other, therefore increasing the difficulty of programming.

2.3 Activation Functions

An activation function handles the value received by the node in an attempt to normalize it, depending of the function we choose, the discrimination frontier can vary significantly. To select the proper discrimination function we must look at the range of values that we are going to be handling.

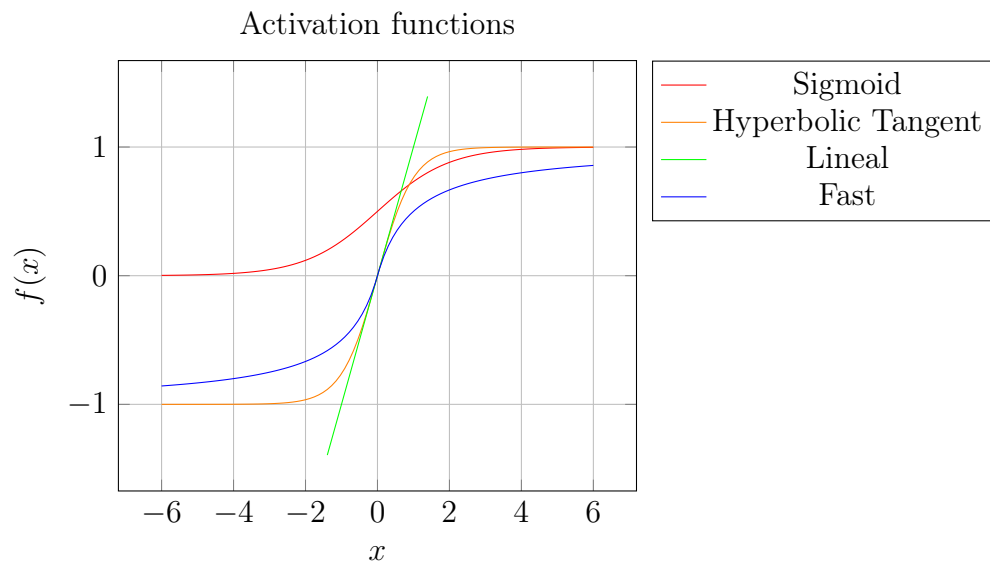
Some examples are:

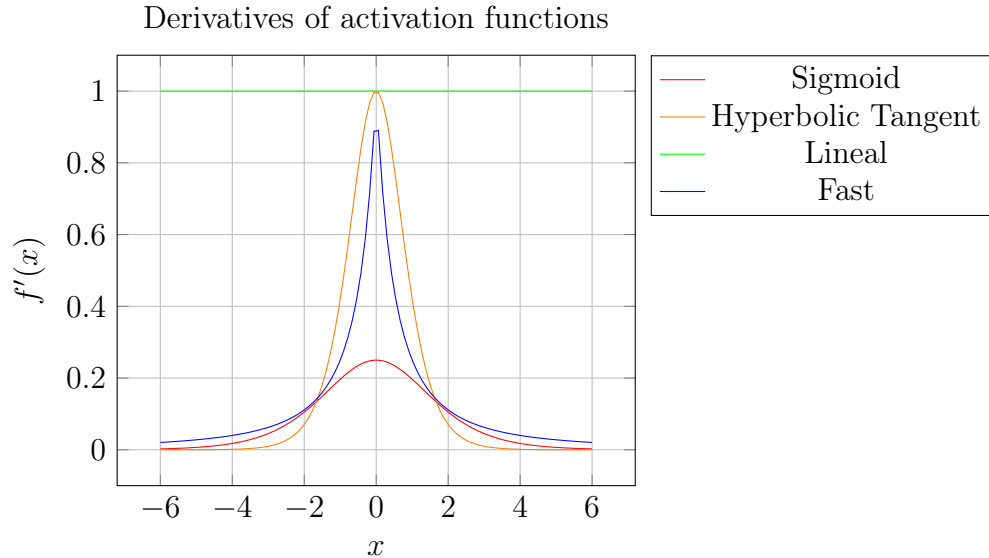
Sigmoid: $f_s(x) = \frac{1}{1+e^{-x}}$

Lineal: $f_l(x) = x$

Hyperbolic Tangent $f_t(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$

Fast $f_z(x) = \frac{x}{1+|x|}$





2.4 Forward Propagation

The process of prediction in a neural network is made through *forward propagation* in this process we will use the entry values $x_n | 0 \leq n < \text{number of input nodes}$ and as a result we will get a series of outputs $y_n | 0 \leq n < \text{number of outputs}$. The process starts by assigning each input value in its corresponding input node (we understand by input node every node inside the input layer).

Now is when the automatic process begins, here we see the pseudo-code:

We begin with L being the total number of layers and l such that $l | 1 \leq l \leq L$ and N_l so that we get the number of nodes in layer l . We are going to use the notation W_{ij}^l for the weight of the connection in layer l from node i to node j in the next layer and V_i^l as the value of node i in layer l . We also start with $2 \leq l \leq L$ & $\forall i V_i^l = 0$ the input nodes V_i^1 will carry the input values.

We will also mention $g(x)$, $g'(x)$ which are the activation function and its derivative, and ρ which is the neural net's *learning coefficient*. We understand that by "learning coefficient" we refer to the importance given to the error in the weight of a connection, a higher learning coefficient will lead to bigger changes when learning, this does not necessarily mean better results; on the contrary, a high learning coefficient normally leads to underfitting.

Foward propagation Pseudo-code

```

for  $2 \leq l < L$ 
  for  $1 \leq n \leq N_l$ 
     $V_n^l = g(\sum_{i=1}^{N_{l-1}} V_i^{l-1} * W_{in}^{l-1})$ 

```

At the end of the algorithm, we will have in $V_i^L | 0 \leq i < \text{Number of ouputs}$; the predicted result based on our input.

2.5 Backward Propagation (Training)

The purpose of this process is to find out how out is every node of the network to get the correct output. Obviously this process can only be done whilst training the neural network. The process target is to minimize the quadratic error at the end of a foward propagation.

$$Quadratic\ error = \frac{1}{No.Outputs} * \sum_{i=1}^{No.Outputs} (\hat{Y}_i - Y_i)^2$$

Here we take \hat{Y}_i as the predicted value of output i and Y_i the value after the foward propagation on node i .

We start with a set of data $S \rightarrow \{x, y\}^n | n > 0$ where $x \in \mathbb{R}^{inputs}$ & $y \in \mathbb{R}^{outputs}$, x being the values inserted to the neural network and y being the expected outputs from that input. We also have new variables ΔW_{ij}^L which will be the change calculated for the weight connecting node i in layer L to node j in layer $L + 1$ and δ_i^L which will be and ρ will be the *learning coefficient*. In this case we will use V^L being an array of values representing the values of the output layer of the neural network.

Single iteration of backProp

```

foreach (x,y) in S
  foward_propagation(x)
  values =  $V^L$ 

  for l from L to 1
    for  $1 < n < \text{Number of nodes in layer l}$ 

```

```


$$\delta_n^l = \begin{cases} g'(V_n^l)(y_n - values_n) & \text{if } l == L \\ g'(V_n^l)(\sum_r \delta_r^{l+1} * W_{ir}^L) & \text{else} \end{cases}$$

for w in  $W_{wn}^{l-1}$ 
 $\Delta W_{wn}^L += \delta_n^l * V_w^{l-1}$ 
for w in W
 $W_{ij}^L += \rho * \Delta W_{ij}^L$ 

```

Lets break down the code:

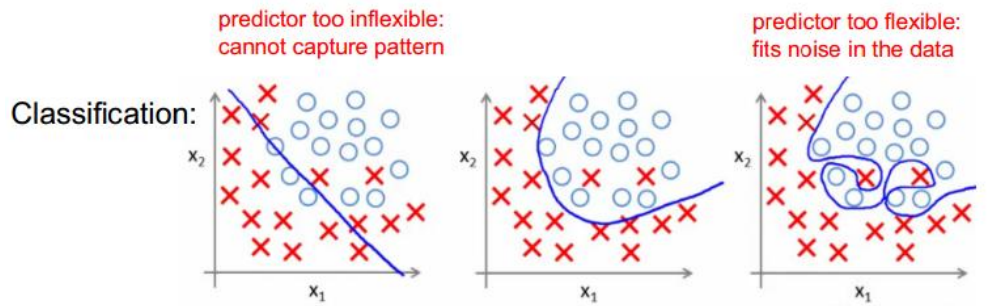
1. For every training sample \mathbf{x} with its expected results \mathbf{y} . We first do the foward propagation algorithm explained before.
2. We then start with the last layer and iterate backwards. And the for every node in the selected layer we calculate its δ .
3. Then for every weight in the selected node we calculate $\Delta \mathbf{W}$ adding the product of δ with the value received from the previous layer.
4. After all the δ has been calculated we get every connection and calculate its new weight using ρ .

Note: This algorithm must be repeated until a stop condition is reached. However depending on how many times we repeat this process and the kind of data we select as our input we might get what its called “overfitting“ or “underfitting“ which means our net has been trained in very specific data, which may misinterpret noise as valid values, or too loosely, not giving a correct decition frontier.

Some “Overfitting“ and “underfitting“ causes.

| | Overfitting | Underfitting |
|----------------------|------------------------------------|--|
| From data | Data too simmilar Repeated data | Few samples Little variance between samples |
| From training | Data too spread | Few iterations Learning coefficient too small |

Visual representation for “underfitting“, correct and “overfitting“ frontiers.



Source: <https://www.prhlt.upv.es/~evidal/students/apr/Tema5/t5aa2p.pdf>

2.5.1 Stop conditions

There are 2 main conditions that make the training algorithm stop:

Convergence on quadratic error

After iterating over the training samples and calculating the quadratic error of the whole sample we then compare the value with the previous iterations. If we find a convergence point where the error doesn't decrease with the same proportion then the learning algorithm will start overfitting.

Maximum number of iterations

This is very rare and very difficult to implement correctly. We establish beforehand a maximum number of iterations. If this number is reached, regardless of the error then the learning stage is immediately halted. Normally this is made to stop very long learning stages. But of course, this can most certainly lead to underfitting.

2.5.2 Data samples

For a further specification of the type of data that should be used to train “correctly“ a neural network, here are some guidelines:

Data quantity

The amount of data has to be big enough to cover most of the general cases but not so big that it contains repeated samples.

Data variation

It is recommended that for each “case“ you have several training samples with a little variation.

Data grouping

You must cover most (if not all) the possible outputs with your training samples.

Chapter 3: The Mathematical Model

3.1 Introduction

Our premise for this project is that the most significant variables are the *distance to the start of the gene* and the *occupation of the genome* at the point of introduction, we also take into account the “neighbourhood“, being the values around the point of insertion of the CRISPR. In this part we’ll explain what this data represent and how to handle them in order to introduce it to our neural network.

When observing the raw data for *Arabidopsis Thaliana* or any genome we can see a collection of millions of genes, which are made of a group of bases, each base can be represented with the letter **A**, **C**, **T** or **G**. Each position, or base, also contains implicitly a value for *nucleosome occupation* and *score*.

3.1.1 Selected data

Gene structure

A gene can be represented with a combination of the letters **A**, **C**, **T** or **G**. This is known as the *FASTA format*, where each letter represents a pair of bases.

Nucleosome occupancy

This represents the “ease by which a CRISPR can be introduced at that position“. The higher value the easier it is to edit the genomic composition with CRISPR/Cas9. There are many other values but these are the relevant to this project.

3.2 The data

After retrieving the data from the web, because there is not full database with the data we want, we are going to create a custom data structure with all the data.

3.2.1 Some file formats

To codify the data into files biotechnological engineers use several standard protocols. These are huge files, over 5 GB in some cases, taking into account that we are only looking at a

very small part of the plant. Here we will overview those that we encountered to create our database. We can find documentation on these formats from:

<https://genome.ucsc.edu/FAQ/FAQformat.html>.

3.2.1.1 The BED format

Browser Extensible Data (BED) format is a file format used by the UCSC genome browser for defining genomic regions. It defines one genomic region (a "BED record") per line. First we have a line (optional) starting with a # with some meta data, separated by commas. Then separated by a line skip we have a line per sequence following this pattern (<https://genome.ucsc.edu/FAQ/FAQformat.html#format1>):

Chromosome <tab> start index <tab> end index <tab> value

Because we are using this format for the nucleosome occupancy we have the chromosome it belongs as the first value. The symbol <tab> represents a tabulator separator.

3.2.1.2 The GFF3 format

GFF is a standard file format for storing genomic features in a text file. GFF stands for Generic Feature Format. GFF files are plain text, 9 column, tab-delimited files. GFF is frequently used in GMOD for data exchange and representation of genomic data. GFF has several versions, the most recent of which is GFF3. GFF3 addresses several shortcomings in its predecessor, GFF2. Its structure is as follows (<http://gmod.org/wiki/GFF3>):

Sequence id

The ID of the landmark used to establish the coordinate system for the current feature.

Source

The source is a free text qualifier intended to describe the algorithm or operating procedure that generated this feature. Typically this is the name of a piece of software, such as "Genescan" or a database name, such as "Genbank".

Type

The type of the feature (previously called the "method"). This is constrained to be either: (a) a term from the "lite" sequence ontology, SOFA; or (b) a SOFA accession number.

Start

Start of the feature.

End

End of the feature.

Score

The score of the feature, a floating point number. As in earlier versions of the format, the semantics of the score are ill-defined. It is strongly recommended that E-values be used for sequence similarity features, and that P-values be used for ab initio gene prediction features. If there is no score, put a “.” (a period) in this field.

Strand

The strand of the feature. “+” for positive strand (relative to the landmark), “-” for minus strand, and “.” for features that are not stranded. In addition, “?” can be used for features whose strandedness is relevant, but unknown.

Phase

For features of type “CDS”, the phase indicates where the feature begins with reference to the reading frame. The phase is one of the integers 0, 1, or 2, indicating the number of bases that should be removed from the beginning of this feature to reach the first base of the next codon.

Attributes

A list of feature attributes in the format tag=value. Multiple tag=value pairs are separated by semicolons.

3.2.2 Extracting the data

Because of the size of the data and the limitations in time for this project we are only going to analyze one of the many chromosomes (Chr1) that *Arabidopsis* contains, each organism contains different quantities of chromosomes.

To acquire these data, we are going to look at different websites that offer this information, with different encodings, for free.

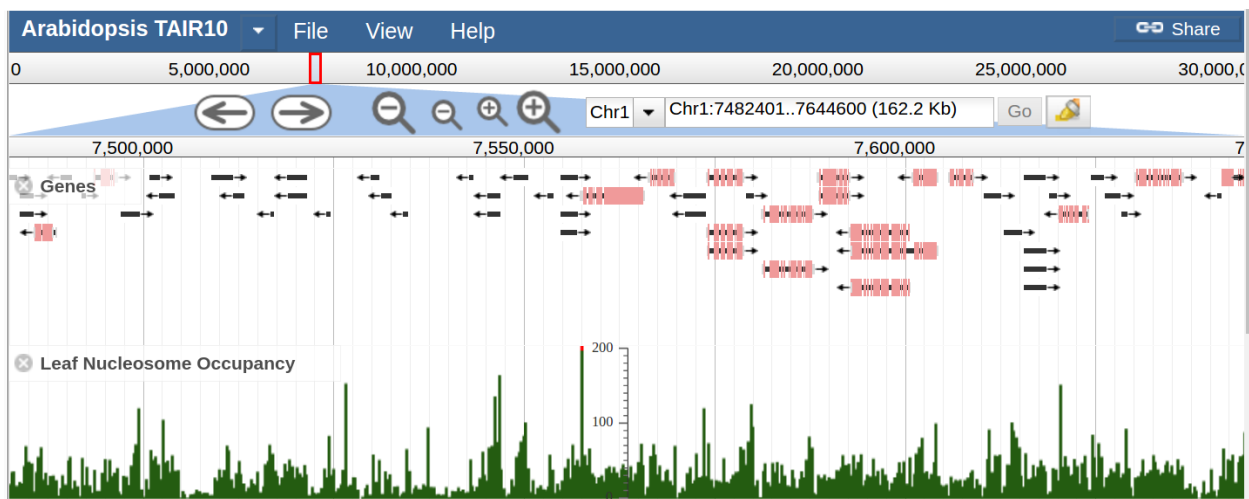
3.2.2.1 Genes

Individual genes are easy to generalize, they consist of a name, start position, end position and orientation. From <http://plantdhs.org/Jbrowse/Arabidopsis> we can find the *GFF3* file for each gene in each chromosome of the plant. With this information we can create a simple data structure to store all the data.

3.2.2.2 Occupancy

Again from <http://plantdhs.org/Jbrowse/Arabidopsis> we can get the *BED* file containing the values for occupancy for each chromosome, these values can vary from 0.0 to 200.0. However this file contains the value for all chromosomes, a quick command to copy all lines starting with “Chr1“ to a different file reduces the file size from 2.9GB to 200 MB. We will use this as a sort of database to check the values in a certain range on the genome. Here we can see how the data is displayed in the online tool for the genes and the nucleosome occupancy that we then download.

Screenshot of genes and nucleosome occupancy in Arabidopsis



Source: <http://plantdhs.org/Jbrowse/Arabidopsis>

3.2.3 Selecting correct insertion positions

We can only insert a CRISPR in places where the bases are **GG**, this means going through the whole structure and selecting all the indexes where this occurrence happens. Because the FASTA file contains only 1 chain of the DNA and therefore some occurrences might occur in the other chain, so following the rules of symmetric fitting we see that **G** binds with **A**, so a regular expression does the trick here as $(GG|AA)$.

3.2.4 Normalizing the data

In order to add some relevance to the surroundings of where we add the CRISPR we are going to also take the next and previous 2000 nucleosome values and normalize them with

a gaussian function. Because we want the highest importance to the closest 200 bases the denominator for the gaussian will be $2 * 100^2 = 20000$

$$gauss(x) = e^{\frac{-x^2}{20000}} \text{ for } x \in [-2000 \dots 2000] = \text{distance to insertion point}$$

$$x \begin{cases} +ve & \text{Opposite direction of gene} \\ -ve & \text{Same direction of gene orientation} \end{cases}$$

This will give relevance to the surroundings of the insertion point progressively smaller the further away and increase the relevance to the closest values, this is very important as we assume that the surroundings take a great role in these type of reactions.

At the end of the full model we end up with an array of size 4000 where position 0 means the furthest value (2000 bases away) in the direction of the selected genes orientation and position 4000 means 2000 bases on the opposite direction. So we en up with:

$$values[i] = gauss(i - 2000) * occupancy[\text{insertion position} - 2000 + i] | i \in 0 \dots 4000$$

3.2.5 Using the neural network

After selecting all the insertion places and normalized each set individually we end up with an *array of values*, where the first values start on one end of the selected region and the last are the other side. We will insert this as is to our neural network and as an output we will get the coefficient between the value of genomic transcription after the insertion of the CRISPR/Cas9 and the value it would have without any insertions. We will give further explanation in the next chapter.

3.2.6 Selecting the best values

After running our algorithm on each insertion point we store the results, select the output with the maximum value and show all of the values in a graph for visual aid.

3.3 Adapting results from humans

Because the experimental results havent been made yet by the laboratory, we took the data from a paper that used CRISPR/Cas9 on human genome (RNA-Guided human genome

engineering via Cas9 by Prashant Mali¹, Luhan Yang, Kevin M. Esvelt, John Aach, Marc Guell, James E. DiCarlo, Julie E. Norville, George M. Church). We also assume that the reactions of the genes when introduced a CRISPR/Cas9 are independant on the medium so it should not affect the fact of changing from plants to humans.

3.4 Results

Because the data set is very small we have chosen the “but one“ strategy to train the data and see how accurate it is.

Basically the “but one“ strategy is selecting one of the samples, training the neural net with the rest of the data and then compare the output of the net with the expected result for that value, this is done for all the values on the set and then the average is calculated.

After the process has finished we end up with just under 92% of accuracy on average (*91.394%*).

Chapter 4: CRISPR/Cas9 Neural Net

4.1 Introduction

Now that we have explained how neural networks work and what problem we are tackling, lets have a look at how to encode the data of the experiment in order for it to be applicable to a neural network.

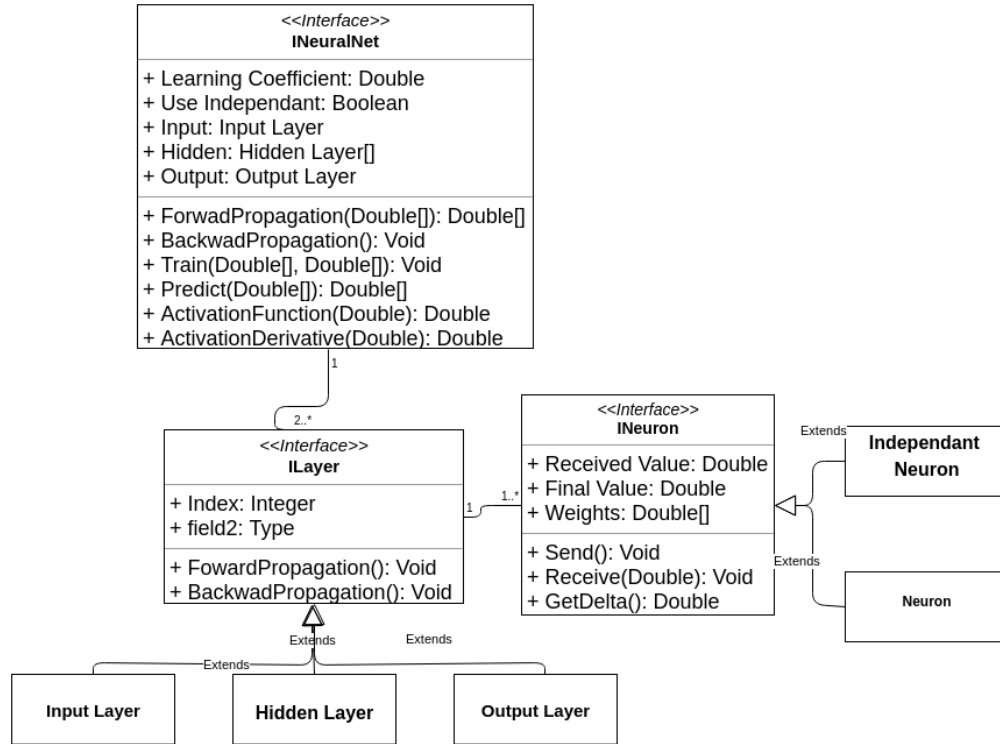
As stated, we must choose:

- An activation function.
- Number of hidden layers with each number of nodes.
- The number of input nodes (and what each represents).
- The number of output nodes (and what they represent).
- A learning coefficient.

Here we will explain how each aspect was chosen.

We have implemented a series of interfaces to be able to adapt to changes in the future as follows:

UML Diagram of implemented ANN

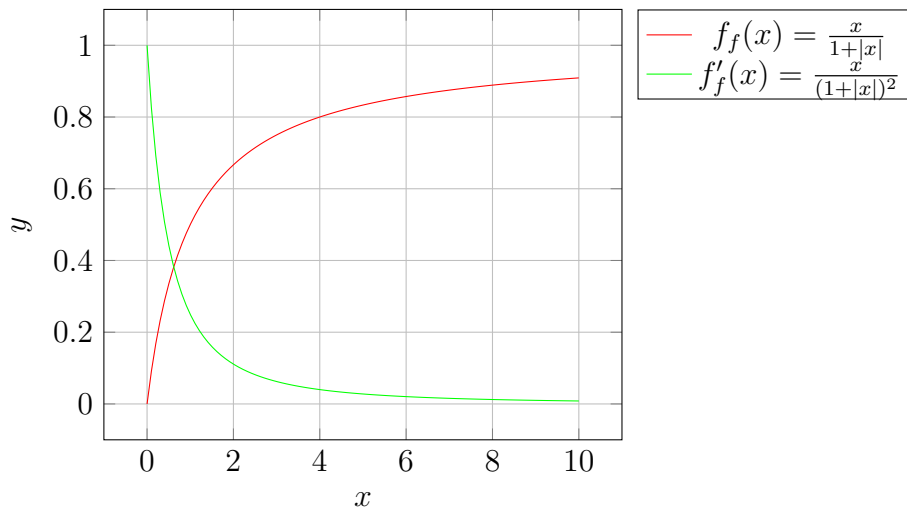


As we can see the *activation function* is implemented as a method inside the *INeuralNet* interface so that when we create a new class that inherits from it we only change that one method and the derivative. We can implement different instances of each interface in case we need to change the structure to a different paradigm.

4.2 Activation Function

Looking at the data that we are going to be handling, we see the range of the values is all the positive real numbers, because of this we select the **fast function** which covers them all, and compared with the sigmoid or the hyperbolic tangent, this gives a smaller gradient for values that are bigger than 0.

Fast function and derivative graph



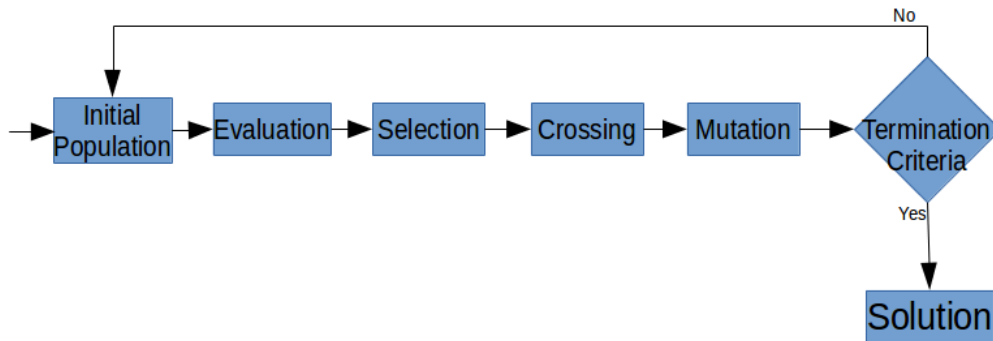
4.3 Handling inputs

The most important part of using a neural network is to understand how we are encoding the input data and what the output data represents. In our case we already explained how we are parsing the data from a custom structure we have called *Gene*, to an array of values. But what do they mean? We can think of the values as influence, the higher the influence the more it affects the reaction at the end. Now the actual position on the array represents the relative position to the beginning of the gene we are inserting it into. With this we are inserting more data into the network, not only the result of the mathematical model but also the real position it is in.

4.4.1 Introduction to genetic algorithms

A genetic algorithm is a search heuristic that mimics the process of natural selection. This heuristic is normally used to generate solutions to optimization and search problems using techniques inspired by natural evolution, such as inheritance, mutation, selection and crossover. Here we can see the structure of a genetic algorithm:

Genetic algorithm flowchart



4.4.2 Our genetic algorithm

We are going to follow the diagram above with the following steps:

Initial population

We will start up with 6 randomly generated neural networks, random numbers between 1 and 20 will be generated for the number of *layers* and 1 to 20000 for the number of *neurons* in each *layer*, each *layer* will have a different number of *neurons*, and one more between 0.1 and 1.0 for the *learning coefficient*.

Evaluation

In this part we need to create a way to compare structures to see which is better. For this, each neural network will be trained with the available data and will be assigned a score based on the number of iterations and the final *quadratic error*.

$$score = quadratic\ error * N^{\circ}\ of\ iterations$$

With this, a smaller score means that a lower error is reached with fewer iterations which means the structure is more efficient, this information is important.

Selection

Here we select the structures we are going to mix (or cross), very basic process, we will group in pairs in ascending order of score, the best score with the second, the third with the fourth and so on.

Crossing

When we have selected all our pairs we are going to create a “crossing“ method which will receive 2 networks and will generate a collection of networks which are the result of crossing the received networks.

We are going to use the *rand()* pseudofunction which generates a random number from 0 to 1 both inclusive, and the generation of the “childs“ will be using the upper and lower bounds defined by the parameters of the networks.

Crossing function pseudocode

```
using A and B as input NeuralNets
upperLayers = max(A.layers.length , B.layers.length)
lowerLayers = min(A.layers.length , B.layers.length)

upperNeurons = max(average_neurons(A) , average_neurons(B))
lowerNeurons = min(average_neurons(A) , average_neurons(B))

upperLC = max(A.Learning_coefficient , B.Learning_coefficient)
lowerLC = min(A.Learning_coefficient , B.Learning_coefficient)

childs = new Collection()

//Create 2 childs between the bounds per LC
childs.add(new NeuralNet(A.input_nodes.Length , A.output_nodes.Length ,
    structure_between(upperLayers , lowerLayers , upperNeurons ,
        lowerNeurons)) , (upperLC - lowerLC)*rand() + lowerLC)
```

```

childs.add(new NeuralNet(A.input_nodes.Length, A.output_nodes.
    Length, structure_between(upperLayers, lowerLayers,
    upperNeurons, lowerNeurons)), (upperLC - lowerLC)*rand() +
    lowerLC)

//Create a child over the upper bounds per LC
childs.add(new NeuralNet(A.input_nodes.Length, A.output_nodes.
    Length, structure_upper(upperLayers, upperNeurons)), (
    upperLC - lowerLC)*rand() + upperLC)

//Create a child under the lower bounds per LC
childs.add(new NeuralNet(A.input_nodes.Length, A.output_nodes.
    Length, structure_lower(lowerLayers, lowerNeurons)),
    lowerLC - (upperLC - lowerLC)*rand())

return childs

```

Assuming we have a constructor of *Neural Net* with the form:

Neural Network constructor

```

NeuralNet(int numberInputNodes, int numberOutputNodes, int []
    structure, double learningCoefficient)

```

And the functions for creating the structure with layers and neurons individually with random numbers between bounds, higher or lower.

After all the childs have been generated they go through the same scoring process as before told and added to the current population.

Mutation

We code that every iteration with a very small percentage of occurrence (under 5%) that some values of some *Nets* are randomly selected and changed, if this happens the mutated selection goes through scoring again.

Purging

After the whole process of generation of childs and mutation is finished we will proceed to eliminate a part of the population in order to ensure we are working with the best

individuals. In this case we will eliminate as many as it takes from the ones with the highest (worst) score until we end up with 6 instances, the same number as the originally generated.

Termination criteria

We determine some very standard termination criteria:

1. Maximum number of iterations, in order for the algorithm to stop eventually.
2. If after the whole iteration the population remains unchanged. Which means we contain only the best individuals and therefore cannot be improved.

If any of this criteria is met the instance with the lowest (best) score is returned. If none of this criteria is met then the algorithm restarts but with the current population until a termination criteria is met.

Chapter 5: Our Application

For this project we have developed a small UI to make it simpler and easier to use. Here we can see how it looks and how its broken down. This application is meant to be used as a reference guide so its use will not be very intensive.

5.1 Roles

We can distinguish 2 different users that can interact with the app, we will refer to them as **researcher** and **manager**. Each with its own priviliges.

5.1.1 The researcher

Basically the *researcher* is a basic user without privileges, the actions it can perform don't alter any data and all he can do is use the application to predict results with the data that is already in the program. This user basically is meant for referential usage and it allows a free use of the app without the fear of cassually "poisoning" the data by introducing an invalid experiment input or a gene with an incorrect setup.

5.1.2 The manager

The *manager* can, on top of all the actions that the *researcher* can do, also alter the data being in the for of adding new genes, or inputing experimental data for (re)training of the neural network.

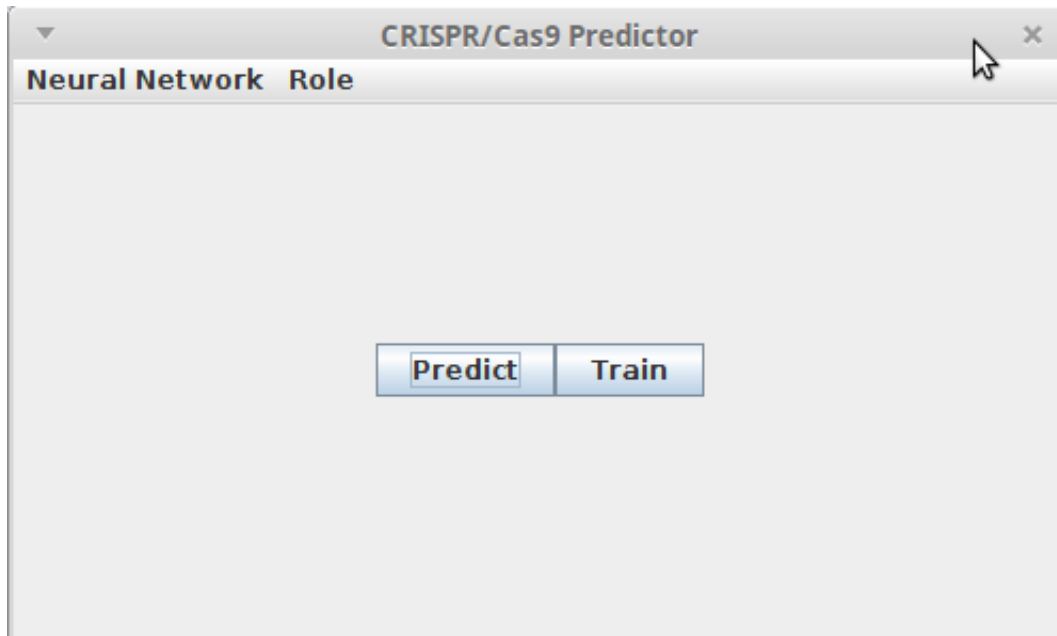
5.2 Overview

We have chosen to stick to simplicity when designing the interface so in the main screen we can select to either train the network or predict a result.

When selected the results will show in the graph below together with a text with the maximum value and position for it.

5.2.1 Main window

Main window



A simple menu bar will be used to switch roles between researcher and a manager, with a simple login with password in order to be a manager and also be able to import/export our neural net structure from a file in case we need a backup or to check results from other users.

The two buttons allows us to either train the network or to predict the best position using the neural network, both buttons open separate windows.

5.2.2 Predict window

Here we introduce the name of the gene we are interested in and the resulting data will appear in the form of a graph and a text showing the value of the best position and its index.

Predict window



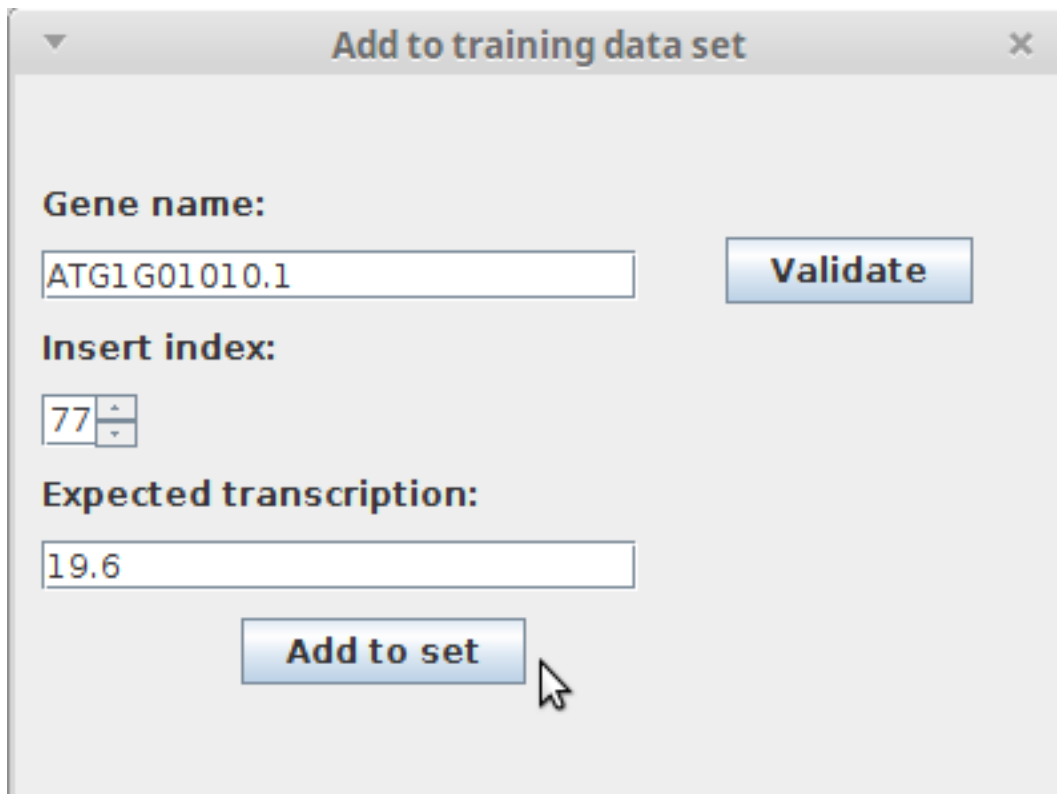
As we can see, some of the indexes have 0 transcription (y axis), which is the proportion index after inserting a CRISPR/Cas9, certain distances from the start of the gene (x axis). This can mean one of two things, either at that position you cannot insert a CRISPR because of a missing *GG* in the genome bases or the actual transcription at that site is 0. There is no way to know with the current setup which of the cases it is, but it can be added easily in the future.

5.2.3 Train window

This window will handle the new values from experimental data are inserted to the training set by the *researchers*.

The *index* and the *transcription value* fields will be disabled until a valid name is inserted and the button of “validate“ pressed. The *index* field will fill automatically with the available insertion places and the *transcription value* field will become active. Perhaps in the future we could adapt it to import a file with several experiment results at once making this process much faster.

Add to training data set window



The screenshot shows a window titled "Add to training data set" with a close button (X) in the top right corner. The window contains the following elements:

- Gene name:** A text input field containing "ATG1G01010.1" and a "Validate" button to its right.
- Insert index:** A spinner box containing the number "77".
- Expected transcription:** A text input field containing "19.6".
- Add to set:** A button at the bottom center, which is being clicked by a mouse cursor.

We got some biotechnology students and professors to test the interface and all of them agreed that the design was very easy to use and would be eager to use the application if it were to be further developed.

Chapter 6: Looking Foward

If in the future we find that the sets of data vary greatly from those that we used there are several ways we can adapt the project to satisfy those needs.

6.1 The Neural Net

We have implemented a simple ANN for this project but in case in the future we need a different structure, like a deep neural network or a recursive neural network the only thing we would need to change would be the *activation function* and the *reference to the next neuron* on the foward propagation as well as editing the format of the backpropagation respectively.

Because of how we implemented the neural net using inheritance only one class is needed to be edited. In the worst case scenarion we create a new class type that implements the needed changes.

For the neural net's structure the genetical algorithm maximizes automatically the neural net when a new training set (experimental data) is introduced so that part is covered.

As for the *Genetic algorithm* to optimize the structure we will reach a point where it take an enourmous amount of time to train every single network so some test must be run in order to see which structure gives the best results with a correct and relatively small data set and keep that as constant.

6.2 The Mathematical Model

If we find other variables that are relevant to the project or that the ones that we used are not the correct ones, the application of our mathematical model is a static function taking as parameters the gene to analyze so we would only need to edit that one function.

Chapter 7: References

- Doench JG, Hartenian E, Graham DB, Tothova Z, Hegde M, Smith I, Sullender M, Ebert BL, Xavier RJ, Root DE. Rational design of highly active sgRNAs for CRISPR-Cas9-mediated gene inactivation. *Nat Biotechnol.* 2014 Dec;32(12):1262-7. doi: 10.1038/nbt.3026.
- Fu Y, Sander JD, Reyon D, Cascio VM, Joung JK. Improving CRISPR-Cas9 specificity using truncated guide RNAs. *Nat Biotechnol.* 2014 Mar;32(3):279-84. doi: 10.1038/nbt.2808.
- Jinek M Chylinski K, Fonfara I, Hauer M, Doudna JA, Charpentier E. A programmable dual-RNA-guided DNA endonuclease in adaptive bacterial immunity. *Science.* 2012 Aug 17;337(6096):816-21. doi: 10.1126/science.1225829.
- Kuscu C Arslan S, Singh R, Thorpe J, Adli M. Genome-wide analysis reveals characteristics of off-target sites bound by the Cas9 endonuclease. *Nat Biotechnol.* 2014 Jul;32(7):677-83. doi: 10.1038/nbt.2916.
- Marraffini LA. CRISPR-Cas immunity in prokaryotes. *Nature.* 2015 Oct 1;526(7571):55-61. doi: 10.1038/nature15386.
- Wu X, Scott DA, Kriz AJ, Chiu AC, Hsu PD, Dadon DB, Cheng AW, Trevino AE, Konermann S, Chen S, Jaenisch R, Zhang F, Sharp PA. Genome-wide binding of the CRISPR endonuclease Cas9 in mammalian cells. *Nat Biotechnol.* 2014 Jul;32(7):670-6. doi: 10.1038/nbt.2889.
- <https://www.prhlt.upv.es/evidal/students/apr/Tema5/>
Multilayer Neural networks - June 2016

Also used but not mentioned in this paper:

- Jose Antonio Climent Hernandez, Valuación de opciones a través de distribuciones sub gaussianas, Editorial Académica Española (Octubre 23, 2014)
- <http://www.sciencedirect.com/science/article/pii/0167779994900485>
Neural-network contributions in biotechnology - June 2016

- <http://www.nature.com/nbt/journal/v16/n10/abs/nbt1098-966.html>
Neural network-based prediction of candidate T-cell epitopes - June 2016
- <http://plantdhs.org/Jbrowse/Arabidopsis>
Online Genome Database - June 2016
- <http://www.genome-engineering.org> Directly at: <http://crispr.mit.edu/>
CRISPR Genome Engineering Resources - June 2016
- <http://cas9.cbi.pku.edu.cn/>
Cas-9 Online Desing Tool - June 2016
- <http://www.e-crisp.org/E-CRISP/designcrispr.html>
CRISPR Online design tool - February 2016
- <http://cbi.hzau.edu.cn/crispr>
- <https://chopchop.rc.fas.harvard.edu/>
Online target site finder - June 2016
- <http://eendb.zfgenetics.org/casot>
- <http://plants.ensembl.org/info/website/ftp/index.html>
Sequence analyzer site - May 2016
- <http://www.biootools.com/col.jsp?id=103>
- <http://www.genome.arizona.edu/crispr>
Rice transposable element database - May 2016
- <http://www.bioconductor.org/packages/release/bioc/html/CRISPRseek.html>
Online CRISPR analyzer - June 2016
- <http://research.microsoft.com/en-us/projects/azimuth/>
- <http://crispr.cos.uni-heidelberg.de/>
CRISPR/Cas9 target online predictor - May 2016