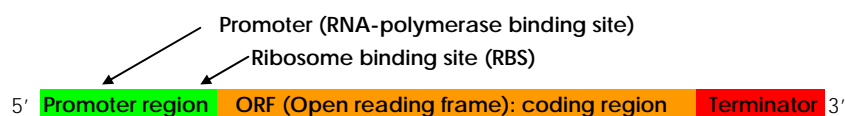


## NEURAL NETWORK LAB #4: DETECTION OF PROMOTER SEQUENCES

### INTRODUCTION

In a genome, a simple (by no means rigorous) definition of genes could be that of self-containing units capable of being read by the genetic expression system and thus translated into proteins. From this definition, it follows that a gene must contain a series of elements that make possible its reading, first by the transcription machinery and later by the translation machinery.



**Figure 1** – Basic structure of a gene. The promoter region contains (at least) two separate elements: the promoter (or core promoter), where RNA-polymerase binds to initiate transcription, and the ribosome binding site (RBS) where ribosomes bind to start translation. The Open Reading Frame is the coding region that ribosomes translate into protein, and the terminator is the region where RNA-polymerase disengages from DNA and stops transcription.

### Gene detection

In silico detection of genes in a newly sequenced genome is often carried out by locating Open Reading Frames (ORF). Their particular ordered nature (they possess Start and Stop codons and must be read 3-modulus wise) and the restrictions that selection imposes on the protein products they encode make them stand out in the genomic background with respect to some statistical measures, facilitating their detection. Gene detection programs use, for instance, codon usage statistics. In a putative ORF read in a correct frame (that is starting from a putative start codon, modulus-3, and ending in a putative stop codon), codons should loosely follow the overall codon usage statistics of that particular organism. If they do, then the putative ORF is probably a valid ORF.

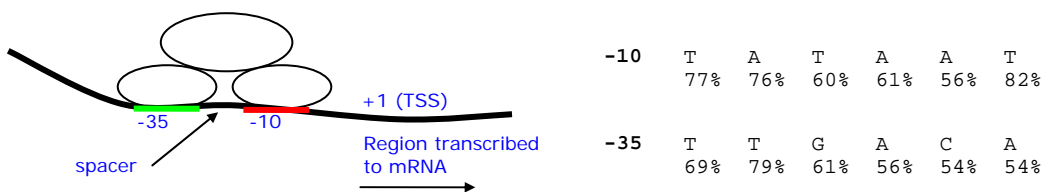
### Promoter detection

Even though gene detection is not simple (e.g. detecting small genes with codon usage measures and the like is not trivial), it is by far much simpler than detection of promoter regions, which are much shorter, do not have fixed metrics and do not stand out significantly from other non-coding (i.e. non-ORF) regions in the genome in terms of statistical measures. In bacteria, detection of promoter sequences is quite easier than in eukaryotic genomes because bacterial promoters are much simpler than eukaryotic ones. Whereas in a eukaryotic promoter RNA-polymerase

can bind at different, loosely-specified sites depending on the presence of other “co-factors” (i.e. other proteins helping RNA-polymerase bind at one site or the other), in a bacterial promoter things are a bit more ordered and RNA-polymerase usually binds to only one site.

### The *Escherichia coli* promoter

In *E. coli*, the binding site for RNA-polymerase (i.e. the core promoter) has been identified for some years, and consists of two binding regions separated by a spacer. The two binding regions are roughly conserved hexamers and are usually called -35 and -10 regions because in the first *E. coli* promoters described they were located at 35 and 10 base pairs (bp), respectively, from the Transcription Start Site (TSS, the position of the first base that is transcribed into mRNA by RNA-polymerase).



**Figure 2** – Basic structure of a bacterial promoter and the consensus sequence for -10 and -35 hexamers.

### GOALS AND SCOPE OF THE ASSIGNMENT

In this lab assignment you will apply unsupervised and (optionally) supervised neural networks to predict promoters in the *E. coli* genome. The assignment consists of two parts. In the first one, which deals mainly with data analysis, you will be first given a dataset of core promoter regions and you will have to analyze it using an unsupervised training regime. Afterwards, you will use also a dataset of control regions and you will evaluate whether the unsupervised network is able to discern between promoters and non-promoters. This task is *mandatory* and should be completed and handed in during the lab session. In the second one, which is *optional*, you will gather the data to train a supervised network to recognize *E. coli* promoters, and you will evaluate their performance on the *E. coli* genome. This second task should be handed in at (or before) the next lab session.

## PART #1: USING SOM TO DETECT PROMOTERS

### Analyzing the training data

We are given a dataset (`E_coli_promoter_collection.txt`) that is said to contain aligned core promoter regions, which we would like to use as our positive training set, but we do not know anything more about it. In order to proceed with the problem of predicting promoters, your first step will be to analyze this dataset with a self-organizing map (SOM) to reveal any hidden clusters (i.e. different classes of core promoters) in our promoter data. To do this, you will have to write a Matlab script that makes use of the `sofm`, `sofmfwd` and `sofmlearn` functions programmed in Neural Network Lab #3<sup>1</sup>.

The script should first load the given dataset and encode the alphanumeric values into something the SOM can process. To load the sequence you can use the `fscanf` function to generate a two-dimensional array of char strings. A standard, but by no means unique, encoding function for DNA sequences is the sparse binary encoding. This encoding technique assigns a four bit (in practice a four byte) value to every DNA base, following a pre-established coding order. For instance, a typical coding scheme could be:

```
A → 0001
C → 0010
G → 0100
T → 1000
```

leaving the 0000 pattern for any character that is not a valid DNA base. Therefore, a DNA sequence like AAGATC would read as 000100010100000110000010 after encoding.

```
%Script to do part#1
%reset loaded variables
clear
%establish limits for sequences
datalines=100;
datalinelen=58;

%load data
mydata=loadseq('E_coli_promoter_collection.txt', datalines, datalinelen);
```

<sup>1</sup> For those who did not complete the SOM functions in the previous lab session, they will be available for download during the lab session.

```
%sparse encode data
for thecount=1:datalines-1
    mycodeddata(thecount,:)=reshape(sparsed(mydata(thecount,:))',1,[]);
end
```

Having loaded and encoded the data, you can then create a SOM with `sofm` and train it with this data as inputs for the SOM (one bit for each input). Since the dataset is not overly long, it would be helpful to program a randomizing function (i.e. a function that swaps pairs of lines of training data in a random fashion) and include it in the `sofmlearn` code, after each epoch has been computed.

```
%create network
mynet=sofm((datalinelen-1)*4,3,'de2','linear');

%train network
mynet=sofmlearn(mynet,datamy,0.1,0.95,0.00001,1,25,1);           %with
randomization at each epoch
```

Having trained the SOM, you should then use `sofmfwd` to analyze to which clusters the network has assigned the input (remember to encode anew the inputs if you have randomized the coded input array during training; else uncoded and coded inputs would not correspond). To do this you can create a cell array with `outputclass=cell(net.nout,1)` and, using a switch-case on the `max_pos` of the SOM output, assign the input pattern to an output class with something like (for case 3) `outputclass{3}=[outputclass{3};mydata(counter,:) ]`.

```
%check classes
class=cell(8,1);
for thecount=1:datalines
    %encode data
    %apply sofmfwd to current input
    %obtain maxpos
    switch maxpos
        %assign vector to class{x}
    end
end
```

With the input data separated into classes, you should be able to analyze the different output clusters in terms of the inputs they group. To do this you can use external software for alignment or information theory analysis of the sequences. You can use, for instance, the CLUSTALW multiple alignment server at <http://www.bioinformatics.nl/tools/clustalw.html>, or the Sequence Logo generator at <http://weblogo.berkeley.edu/logo.cgi>. In the former case, you should

input the sequences in multi-FASTA format, with a >line with a unique identifier before each sequence:

```
>1
GACACCATCGAATGGCGCAAAACCTTTCGCGGTATGGCATGATAGCGCCCGGAAGAGA
>2
TGTGCAGTTTATGGTTCCAAAATCGCCTTTTGCTGTATATACTCACAGCATAACTGTA
```

### Evaluating SOM performance

To evaluate the performance of SOM to detect promoters, we are given another set with 1500 sequences corresponding to non-coding & non-promoter fragments of the *E. coli* genome (`E_coli_negative_set.txt`). With this new set and the former set of core promoter regions (our positive set), we will create a new training dataset consisting of interspersed positive and negative examples. Since there are 15 times more negative examples, we will create the new dataset by sequentially reading negative set examples and getting random examples from the positive set, so that our final training set follows always the same pattern:

```
negative #1
positive (random sample)
negative #2
positive (random sample)
...
```

With this training set, we will train our SOM network and we will analyze its output distribution in the same manner as we had done previously for the positive set. This time, we are interested in the proportion (size) of negative and positive samples that get clustered in the same class. For the SOM to work well as a detection system, the output classes holding positive examples should hold as few negative examples as possible.

### Evaluation and delivery

To complete this part of the assignment, you should be able to display, demonstrate and explain the scripts and the routines you have programmed, namely:

- Training data analysis script / SOM detection script
- function [sequences] = loadseq(filename, linelength, seqlength)
- function [codedseq] = sparsed(seq)
- function [mymatrix] = randomize(mymatrix, cycles, lines)



## PART #2: USING MLPs TO DETECT PROMOTERS

In this *optional* part you shall use MLPs to implement a promoter detection system. First, you must download and save a collection of aligned core promoter sequences from the web page of Mike O'Neill at the University of Maryland Baltimore County (<http://www.research.umbc.edu/~moneill/>; download the prom75.seq collection). This will constitute your positive training set. Secondly, you have to generate a negative dataset. You can do this by generating random sequences or by extracting random sequences from the *E. coli* genome, but you must justify which strategy you use.

Finally, you have to create a MLP network and train it with both sets. After training the network, you must download the *Escherichia coli* plasmid pMUR050 sequence as your validation set. This is the complete sequence of a plasmid (like a genome but smaller) that confers *E. coli* cells (if they acquire the plasmid) with resistance to a bunch of antibiotics. So that you know, the isolates for sequencing this plasmid come from the *Facultad de Veterinaria, Universidad Complutense de Madrid, Sanidad Animal, Spain, Madrid*, where multi-drug resistant *E. coli* cells have been isolated (i.e. this insidious bug could be lurking right now in your bathroom). You must then write a function that scans the *Escherichia coli* plasmid pMUR050 sequence base-by-base, using a sliding window of 59-bp, encodes that 59-bp window and uses it as input for the MLP. You only have to scan one strand (whichever of the two you prefer). You will save the MLP outputs in a vector and then use a threshold to determine at which positions the MLP has detected a promoter.

To download the *E. coli* genome sequence (or the *Escherichia coli* plasmid pMUR050 sequence), go to the GenBank database at <http://www.ncbi.nlm.nih.gov/> and search for *Escherichia coli* in the Taxonomy category. A list of *E. coli* serotypes<sup>2</sup> will follow. Choose the main *Escherichia coli* (*Click on organism name to get more information*) and then *Genome sequences*. For any given genome/plasmid sequence you choose, you will have a reference sequence (*RefSeq*) which you can display as GenBank or FASTA. The GenBank format (default) gives a detailed view of the sequence, with fields identifying each element

(typically genes), the position they are at, their orientation, the proteins they encode, etc. and the numbered sequence at the end of the file. The FASTA format gives only the raw sequence, preceded by a single >line at the beginning of the file. With the GenBank sequence you can identify where genes are and, therefore, infer if promoters could be lying there (if the space between consecutive genes in the same orientation is smaller than 30 bp you can consider that they are in an operon configuration and, therefore, estimate that only the first gene carries a promoter). The FASTA format is useful to get the sequence and manipulate it (remove carriage-return characters, extract 59-bp chunks, etc. You can do this with a custom C program or using a decent text editor, like UltraEdit).

### Evaluation and delivery

The report on this optional part should be handed in before (by email) or at (in hand) the next lab session. The *short* report should include the programmed code, explain briefly what you have done, how and why, and it should also address specifically the following questions:

- How did you generate your negative set and why? Do you think it can affect the network efficiency at predicting promoters?
- Using the threshold you esteem best, how many positives do you obtain? How do the results vary depending on the threshold? How many correspond to real promoters (insert a table with the network predictions you think are true, the distance of the hit from the ORF the promoter is supposed to control and the name of the gene [*gene/locus\_tag*]<sup>3</sup>)? Do you obtain sometimes more than one hit per promoter region? If so, what do you think that could mean? What is your rate of false positives? Do you think it is acceptable?
- Do you have any ideas as to how you could improve the results?

---

<sup>2</sup> The standard *E. coli* serotype (the reference lab *E. coli*) is *Escherichia coli* K-12.

---

<sup>3</sup> You can obtain a table of *protein coding* genes from the genome page of *Escherichia coli* plasmid pMUR050 at NCBI.