FIGURE ANALYZER: A tool for image recognition

Bernardo Rodríguez, Gerard Martínez & Adrià Aterido.

Master in Bioinformatics. University Pompeu Fabra

Introduction

Figure Analyzer is an implementation of the spatial pooler function from the Hierarchical Temporal Memory (HTM) Cortical Learning Algorithm described by Numenta (www.numenta.com), a neural model to simulate a natural function of the neocortex: image recognition. An Naïve Bayes classifier was added to the spatial pooler to make it able to recognize a query image classifying it into a kin.

The tool is provided in two different ways: one with a terminal interface and another with a more user-friendly graphical user interface. In both cases, it takes as input several sets of learning images (i.e. a set of different kind of squares and another of crosses), generates a sparse distributed representation of them and builds a probabilistic model. After that, a query-image is asked to be introduced in order to be recognized. Finally, Figure Analyzer will ask the user what he wants as output: the matrices generated in the activation and inhibition process that takes place in the spatial pooler function or the results of figures recognition process, which are the probability of this figure to belong to each one of the learning sets.

Methods

- HTM Cortical Learning Algorithm

Figures Analyzer is an implementation of the spatial pooler function from a HTM algorithm, a neural model to simulate a natural function of the neocortex: image recognition. It reads an image in form of a binary matrix and generates a sparse distributed representation of the original input. This is achieved by (1) dividing the original matrix in receptive fields, each of them corresponding to a so-called column, (2) declaring as active columns those with enough positive input and (3) carrying out a process of lateral inhibition in which only the most active columns are kept active. This last step expects to stimulate the obtention of more sparse representations. Also, notice that in each of the last steps, arbitrary thresholds based on statistics were used by default but they can be changed by the user anytime.

Furthermore, a last step was implemented in the code and it is the one of learning. This implies that several images of each object type have to be provided in order to develop the machine-learning feature. This way, after each learning cycle, (1) the algorithm provides an easy way to strengthen repetitively (common) activated synapses, understood as every cell of a column receptive field, to establish and empower certain sparse distributions for each type of object. On the other hand, (2) an activation boost factor is positively corrected for less or never activated columns in order to become over excitable and induce them to be part of new sparse distributions after future new objects input. By doing so, we can easily obtain notably differentiated sparse distributions for each type of object and thus ease the task of object recognition.

- Naïve Bayesian analysis

Figures Analyzer provides an additional feature to recognize a provided query image after the program has learned several sets of learning images (i.e. a set of different kind of squares and another of crosses). It is a Naïve Bayes classifier, these kind of approaches are proved to be really useful to solve some biological problems as the prediction of RNA binding sites in proteins.

It uses several learning sets of images to build a probabilistic model. It contains the probability of each column to be activated and non activated in each one of the learning sets. In order not to have any column with a probability of been activated of 0 the Laplace estimator (**Eq. 1**) was used, which adds 1 occurrence to every possible attribute value .

Eq. 1)

$$P(a) = \frac{n_a + 1}{n + m}$$

Where **n** is the absolute frequency of the **a** state and **m** is 2 since there is two possible attribute values: active and not active.

Once the model is built it is able to recognize a query sequence (following the example, a square or a cross). It is done by using the previously computed probabilities and the Naïve Bayes equation (**Eq. 2**).

Eq. 2)

$$V_{NB} = arg \max_{v_j \in V} P(v_j) \prod_{r=1}^{n} P(a_r | v_j)$$

It means that the probabilities of the query image to belong to each one of the learning sets are computed, compared and the bigger one is taken as the solution to the recognition issue. This probabilities were computed as neperian logarithmic probabilities since they are really low probabilities and they could generate internal computational problems.

Finally, a threshold of -12 is considered to do the image recognition. If the best score given is not over the threshold it is not good enough to do a good recognition so it lets the user to know that no good prediction has been done. The score value for an image generated randomly is -25 (ln 0.5^36), so we decided to set a higher score of -12 to be sure with a probability of more or less 80% that the prediction done is good.

Although this is not a fully biologically-coherent mechanism, it provides an easy solution to prove that the learning procedure is in fact working.

Implementation

The program has been implemented in python language and it is a clear example of object-oriented programming. Each of the columns of the system is in fact a column object and, furthermore, each of them has an array of objects synpases embedded as an attribute. This organization really facilitates the job in every possible level, such as inhibition, update of columns/synapses strengths or even in the creation of weight matrixes. The main algorithm and a brief description of the most important functionalities are detailed next.

First of all, given an initial image dimension (square of a given *len_side*), the column objects and the respective synpases will be created. Each of the synapses will be given an initial random value of permanence around a provided threshold (global variable *connectedPerm*). By doing so, we will obtain a random number of synapses that will result connected to its corresponding column.

Now, for each of the images provided during the learning phase, the matrix will be read and stored. Thus, all the images of a certain object type (e.g. squares) will be available to be cyclically “shown” to the learning-machine a specified number of times (global variable called *learning_cycles*).

In each of these image exposures, the program will call the correct stored image matrix and divide it in receptive fields, each of which corresponds to a certain column. These submatrixes will be passed to a secondary subroutine which will analyze its content and compute the activity state for each synapse (with its corresponding cell on the input matrix). The synapse will become active only if its own value of permanence is higher than the threshold *connectedPerm* (which would mean that the synapse is connected to the column) and if the matrix has a positive input in that particular position (a 1 instead of a 0). Finally, a column attribute called *overlap* will be updated with the sum of activated synapses and multiplied by the factor *boost*, which is 1 by default but will be increased in some cases later explained. At this point, each column will be assessed if the number of active synapses is enough in order to activate the whole column. This will happen if the *overlap* is higher than a specified global threshold *minOverlap*.

Right after, the inhibition process will start. For each column, a simple function based on a mathematical algorithm will return a list of all the neighboring columns. The neighbor columns, then, will be sorted by its value of overlap and the global threshold *desiredLocalActivity* will decide which is the minimum value of *overlap* that a column must have in order to not be inhibited by a neighbor one.

Later, the process of learning will take place. The permanence of each activated synapse will be increased by a global *permanenceInc* value, and decreased by p*ermanenceDec* the one of those not activated. On the other hand, it will update the duty for each column and, if necessary, its *boost*. The duty is a measure of how many times a column has been activated. For each time a column is activated, a 1 is appended to the array *activeDutyCycle.* If its inhibited or not activated, a 0 will be appeneded. For each cycle a duty sliding average (whose length is determined by the global variable *sliding_average*) will be calculated for every column and will be compared to the 1% of the highest neighbor duty average. If the own average is lower than this 1% highest neighbor average, the column boost will be increased by a global value of *boostInc*.

In synchrony with the algorithm explained, every time a column is activated, an attribute of the column called as the name of the object type (eg. cross, square), will be added plus one in order to create the weight-like matrix used afterwards to make the recognition.

At the end of the learning cycles, an external image will be asked to be provided. The image will be read and the activation/inhibition state of the columns will be assessed. Then, the sum of logarithms of the frecuencies of 1 or 0 will be done for each object type available. A result with the best scoring object type will be displayed as STDOUT or as a pop-up window. In case that the score is under a threshold, no good guess will be displayed.

How to use

The program is provided in two different ways: one with the terminal interface and another with a more user-friendly GUI.

On the one hand, to use the terminal version one must type “python”, the name of the program and provide as arguments the name of the folders that we want the program to get trained with (each one containing a single type of similar objects, e.g. cross, square). At the end of the learning cycles, the user will be asked to type the name of an image to recognize. After pressing intro, the scores and the best guess will be displayed as STDOUT.

On the other hand, to use the graphical interface one must type “python” followed by the name of the program. Automatically, a pop-up window will be displayed to briefly explain the purpose of *Figure Analyzer* software. This kind of interface is really user-friendly because each step is carefully detailed on the screen through pop-up windows. The following message that will appear is just to clarify all the steps that will be carried out by the user. After these informative two messages, the user will be asked to indicate the number of learning cycles that will be applied to the learning-machine and how many shapes will be used for learning (corresponding to the number of image-directories). Then, each directory will be selected easily using the graphical interface. Right after, the program starts running in order to learn. When *Figure Analyzer* has learned, the user can choose whether to start the analysis of the query image recognition or not. In the positive case, the user must select the query image and which output would like to display through new pop-up windows. Finally, the image recognition results will be displayed on the screen if the user selects “Image recognition” or a results file will be saved in the chosen directory with the desired name if the user selects “Results file”.

Contribution

Although we did most of the work together, if we are asked about the personal contributions or special motivation this would be an approximate but inaccurate work distribution:

- Bernardo: classes definition, extract data from the input and part of weight matrices.
- Gerard: inhibition, part of learning and part of weight matrices.
- Adrià: part of learning, part of weight matrices and graphical user interface.