Metaheuristics for Optimization

Series 5 : Network learning using PSO

Part 1 : Return no later than November 14, 2022 (16h00)
Part 2 : Return no later than November 21, 2022 (16h00)

This exercise is intended both as an application of the Particle Swarm Optimization (PSO) algorithm, and also as an express introduction to the theory of artificial neural networks.

More precisely, the goal of this series will be to use the PSO algorithm to learn the weights of a Neural Network Model able to classify hand-written digits.

1 Artificial neural networks

Artificial neural networks have been introduced in the 50's as a model which is not really supposed to model the brain in a realistic way, but rather to mimic some of its most striking characteristics by aggregating elementary building blocks, which are themselves designed as a rough model of real neurons.

An *artificial neuron* can be seen as a cell which applies some non-linear operation to its input and returns as an output the result of this operation. The non-linear operation is often a *sigmoid* function $g(z) = \frac{1}{1+e^{-z}}$.

A *neural network* is thus constructed by superposing layers of such neurons. The neuron of each layer takes as input a *weighted* sum of the previous layer's outputs. The advantage gained by considering several neurons is that we can then simulate complex functions even if individual neurons perform simple, though non-linear, operations. In simple words, a neural network takes inputs, processes them in a quite intricated way through successive layers and, spits out an output.

Artificial neural networks are widely employed in pattern recognition, and this is precisely the task we want to complete in this exercise. Indeed, pattern recognition requires presenting a picture (input) to a device which is then supposed to return an output like 'yes, this picture depicts a rabbit', etc. You will be given a set of 200 pictures (400 pixels each) that you will have to show to your network.

Of course, in order for a pattern recognition device to be able to do the work it has been designed for, we first have to train it to. This amounts to choosing the most appropriate *weights* of our Neural Network Model, and this is where the PSO algorithm can help us.

2 The Particle Swarm Optimization algorithm

The *Particle Swarm Optimization (PSO) algorithm* is a biologically-inspired algorithm motivated (somewhat similarly to the Ant algorithm we studied in the last exercise) by the fact that groups of individuals tend to be more efficient in finding solutions compared to isolated individuals. Individual behaviour, nonetheless, keeps some importance and finding a good PSO is a fairly subtle balance between *individuality* and *sociality*. This is implemented as follows in the PSO algorithm.

Let us assume we deal with a set of N particles and denote by $\boldsymbol{s}_t^i$ and $\boldsymbol{v}_t^i$ the position and velocity in the research space of the i-th particle at iteration t. Let us further denote by $\boldsymbol{b}_t^i$ the current best position of particle i and $\boldsymbol{b}_t^G$ the current *global* best position, in the sense that we consider all positions of all particles so far. The velocities and positions are then updated as follows :

$$\boldsymbol{v}_{t+1}^i = \omega \boldsymbol{v}_t^i + c_1 r_1 (\boldsymbol{b}_t^i - \boldsymbol{s}_t^i) + c_2 r_2 (\boldsymbol{b}_t^G - \boldsymbol{s}_t^i) \quad (1)$$

$$\boldsymbol{s}_{t+1}^i = \boldsymbol{s}_t^i + \boldsymbol{v}_{t+1}^i \quad (2)$$

with random initial conditions. ω is inertia constant ; c_1, c_2 are cognitive and social parameters ; and r_1, r_2 are random numbers between 0 and 1. In addition to the velocity update equation (1), the particles' velocities should not become arbitrarily high. For this purpose, we apply a cut-off v_{max} on the absolute value of each velocity component. The stopping criterion can be fixed as a maximal number of iterations t_{max} or a desirable value of the fitness function.

In plain words, expression (1) means that the i-th particle is attracted towards the best solution it has visited so far (second term), which accounts for individuality, but also towards the best solution ever visited by the full swarm (third term), which accounts for sociality. A pseudo-code is proposed below.

Algorithm 1 PSO algorithm

Initialize
t = 0 ;
Initialize randomly the positions $\boldsymbol{s}_0^i$ in the domain to be explored and the velocities $\boldsymbol{v}_0^i = 0$ for all particles $i = 1..N$.
Do
  For each particle :
   Calculate its fitness J_t^i ;
   If $J_t^i \leq J_{best}^i$ then $J_{best}^i = J_t^i, \boldsymbol{b}_t^i = \boldsymbol{s}_t^i$
  End for
  Calculate $J_{best}^G = min_i J_{best}^i$, update $\boldsymbol{b}_t^G = \boldsymbol{b}_t^{arg(min_i J_{best}^i)}$
  For each particle :
   Randomly generate r_1, r_2
   Update particle velocity by formula (1)
   Update particle position by formula (2)
  End for
  t = t + 1 ;
Until (end criteria are met)

3 Our network

The neural network we will use is formed by three layers of neurons. It is similar to the neural network displayed in Figure1, except that ours has only one output unit.

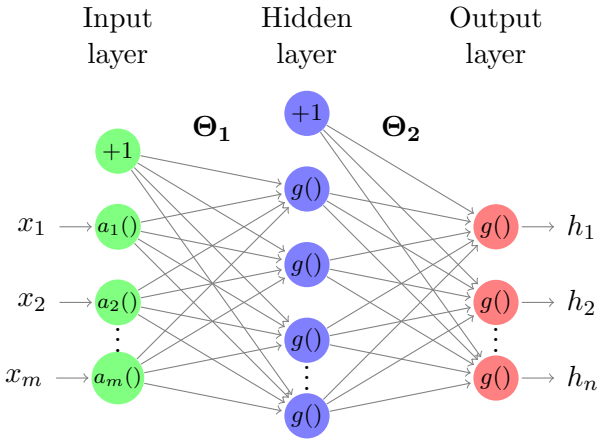

FIGURE 1 – A 3-layers neural network. This one has several outputs at the last level, while the one considered in the exercise has only one. In our case we also set $a_i(x_i) = x_i$.

The first layer contains 400 neurons whose role is to 'download' the 400 pixels of the picture input, plus one neuron taking value +1 giving some bias to the network. Each of these 401 neurons now passes their output to the second layer.

The second layer contains 25 neurons, plus one neuron for the bias (see figure). The task of these 25 second layer's neurons is now to compute the sigmoid of the weighted sum of the first layer's outputs. In more mathematical terms, a vector of size 401 is first transformed linearly into a vector of size 25, through a matrix multiplication $v \to \mathbf{\Theta}_1 v$, where $\mathbf{\Theta}_1$ denotes a matrix of size 25×401, then the sigmoid g is applied on $\Theta_1 v$.

The third layer contains only one neuron receiving the weighted sum of the second layer's outputs ($25 + 1$ outputs) ; so here again, we are dealing with a matrix multiplication of the form $v \to \mathbf{\Theta}_2 v$ with $\mathbf{\Theta}_2$ a 1×26 matrix, then the sigmoid g is applied on $\Theta_2 v$.

In summary, if we denote by $\boldsymbol{x}_k$ the vector formed by the pixels of the k-th training picture, the output we get when presenting this picture is given by

$$h_{\mathbf{\Theta}_1,\mathbf{\Theta}_2}(\boldsymbol{x}_k) := g\left(\mathbf{\Theta}_2 \cdot [1, g(\mathbf{\Theta}_1 \cdot [1, \boldsymbol{x}_k]^T)]\right) \tag{3}$$

or with a more explicit notation,

$$\boldsymbol{x_k} = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix}$$

$$\boldsymbol{z} := \boldsymbol{g}\left(\underbrace{\begin{bmatrix} \theta^{(1)}_{1,1} & \cdots & \\ \vdots & \ddots & \\ & & \theta^{(1)}_{s,(m+1)} \end{bmatrix}}_{\boldsymbol{c}} \cdot \begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_m \end{bmatrix}\right) = \begin{bmatrix} g\,(c_1) \\ \vdots \\ g\,(c_s) \end{bmatrix} = \begin{bmatrix} z_1 \\ \vdots \\ z_s \end{bmatrix}$$

$$h_{\boldsymbol{\Theta_1},\boldsymbol{\Theta_2}}(\boldsymbol{x}) := \boldsymbol{g}\left(\begin{bmatrix} \theta^{(2)}_{1,1} & \cdots & \theta^{(2)}_{1,(s+1)} \end{bmatrix} \cdot \begin{bmatrix} 1 \\ z_1 \\ \vdots \\ z_s \end{bmatrix}\right)$$

$$y^{\text{predicted}}_{\boldsymbol{\Theta_1},\boldsymbol{\Theta_2}}(\boldsymbol{x}) := \begin{cases} 1 & h_{\boldsymbol{\Theta_1},\boldsymbol{\Theta_2}}(\boldsymbol{x}) \geq 0.5 \\ 0 & h_{\boldsymbol{\Theta_1},\boldsymbol{\Theta_2}}(\boldsymbol{x}) < 0.5 \end{cases}$$

where $y^{\text{predicted}}_{\boldsymbol{\Theta_1},\boldsymbol{\Theta_2}}$ is supposed to tell us *what this picture is*, and the vector $h_{\boldsymbol{\Theta_1},\boldsymbol{\Theta_2}}(\boldsymbol{x})$ represents the state of activation of the output layer of neurons.

Knowing the label y_k of each training picture x_k, a measure of how successful our recognition algorithm is provided by calculating the fitness function

$$J_k(\boldsymbol{\Theta}_1, \boldsymbol{\Theta}_2) := (y_k - h_{\boldsymbol{\Theta}_1,\boldsymbol{\Theta}_2}(\boldsymbol{x}_k))^2 \tag{4}$$

on every training picture we presented. If there are m of them, we will aim at minimizing the overall fitness

$$J(\boldsymbol{\Theta}_1, \boldsymbol{\Theta}_2) := \frac{1}{m}\sum_{k=1}^{m}(y_k - h_{\boldsymbol{\Theta}_1,\boldsymbol{\Theta}_2}(\boldsymbol{x}_k))^2 \tag{5}$$

*

In this exercise, you are provided a file *X.dat* containing 200 rows each corresponding to a training picture and displaying the grey values of the 400 pixels. Each picture displays a handwritten '2' or '3' (See Figure2). We also provide another file *Y.dat* containing 1 if the picture represents a '2' and 0 if it is a '3'.

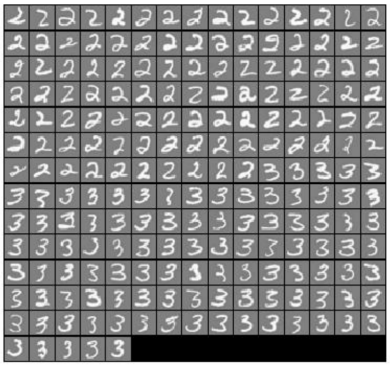

FIGURE 2 – 200 training pictures.

4 Work to do

4.1 Part 1

Work on implementing the functions that will allow you to :
— Get sigmoid(x)
— Calculate the state of activation of the output layer of neurons
— Calculate the overall fitness of a particle
— Calculate prediction accuracy
— Implement the PSO algorithm [1] in order to find the weight matrices $\boldsymbol{\Theta}_1, \boldsymbol{\Theta}_2$ minimizing the fitness (5).

To start, you can take advantage of the given skeleton code.

4.2 Part 2

1. Run the PSO algorithm for different N (number of particles). How does the number of particles affect the results of the PSO algorithm ?
2. Run the PSO algorithm for different cutoff velocity v_{max}. How does the velocity cut-off v_{max} affect the results of the PSO ?
3. Run the PSO algorithm 10 times and report the optimal value $J(\boldsymbol{\Theta}_1, \boldsymbol{\Theta}_2)$ you get each time. In case t_{max} differs between runs, plot the best fitness for each run vs. number of iterations it took.
4. For *one* of these runs, plot $J(\boldsymbol{\Theta}_1, \boldsymbol{\Theta}_2)$ as a function of the iteration number.
5. For each run of the PSO algorithm, calculate and report the *prediction error* $R := \frac{1}{m} \sum_{k=1}^{m} |y_{predict}(\boldsymbol{x}_k) - y_k|$ for the obtained solution $(\boldsymbol{\Theta}_1, \boldsymbol{\Theta}_2)$, where we define $y_{predict}(\boldsymbol{x}_k) := 1$ if $h_{\boldsymbol{\Theta}_1, \boldsymbol{\Theta}_2}(\boldsymbol{x}_k) \geq 0.5$ and 0 otherwise.

In the comments, answer the following questions :

1. What is the Search Space of our problem ?
2. Explain the coefficients c1 & c2
3. How is PSO similar to Ant Algorithm ?

5 Work to Return

For this series, submit your code, results, and comments, and upload them on moodle, by :
— Monday, November 14, 2022 at 16h00 for Part 1.
— Monday, November 21, 2022 at 16h00 for Part 2.

The use of python notebooks is highly recommended, to be able to include code, results, and comments in the same file. Graphs must include a title and a legend for the axes when needed.

1. Put $c_1 \approx c_2 \approx 2$. Choose ω slightly smaller than 1 (ex. 0.9). You are free to choose the number of particles and the stopping criterion, ex : value to assign to t_{max}, but your choices have to be explained.