Stochastic Optimization: Project 1

Vittoria Civiero, Filippo Grobbo

June 2022

Introduction to the problem and theoretical aspects

In the Reed-Frost model we firstly simulate the number of infected, susceptible and removed over a population in order to find the optimal level of containment measures that minimizes costs. In particular our purpose is to optimize the total average cost given by the sum of the health system and containment. These costs are affected by the probability of one to one infection, p. Given a vector of 11 values of p we simulate 1000 repetitions of the Reed-Frost process for 60 consecutive days. By training a Neural Network we expect to find the optimal value of p that minimizes the above total average cost.

Non-linear Neural Network

We implemented one Matlab script and two functions to approach this problem. The first one, called project1test, contains the definition of variables and the simulations of infected and susceptibles. These latter follow respectively the updating rule:

$$I_{n+1} \sim Bin(S_n, 1-(1-p)^{I_n})$$

$$S_{n+1} = S_n - I_{n+1}$$

$$S_0 = 999, I_0 = 1$$

After that we compute the total average cost as the mean of the total number of infected during each simulation plus the containment cost given by the formula: $c(p) = \left(\frac{0.003}{p}\right)^9 - 1$. Subsequently, we call the function nnbackprop and nnpredict where backpropagation and the prediction of final results are implemented. The first one is used to train the neural network, while the second to find the best possible p in [0.005,0.03] which minimizes the total average cost.

The principles of the first algorithm can be summarise in the following pseudocode:

Algorithm 1 nnbackprop

function nnbackprop(xp,yavcost,mu,H)

Input

xp: probability of one to one infection
yavcost: corresponding total average cost;
mu: stepsize for the steepest descent method
H: number of hidden nodes

Output

o: output obtained from the training of the neural network
xh: weights of the linear combination between hidden and output layer
wih: weights of the linear combination between input and hidden layer
SSE: sum of squares error between o and yavcost
b: bias node

Standardize the input xp and yavcost

Set $w(i,h), x(h), b$ to small random numbers between 0 and 0.5 $\quad \forall\, i,h$

Compute $v_p^*(h) = \sum_{j=1}^{I} w(j,h)u_p(j) \quad \forall\, h,p$ (I= $\#input\ nodes$)

Compute $v_p(h) = \dfrac{1}{1+exp(-v_p^*(h))} \quad \forall\, h,p$ (sigmoid function)

Compute $o_p = b + \sum_{h=1}^{H} x(h)v_p(h) \quad \forall\, p$

Update $b = b + \mu \sum_{p=1}^{n} (y_p - o_p)$

Update $wih(i,h) = wih(i,h) + \mu \sum_{p=1}^{n} (y_p - o_p)v_p(h)$

Update $xh(h) = xh(h) + \mu \sum_{p=1}^{n} (y_p - o_p)v_p(h)$

Compute $SSE = \sum_{p=1}^{n} (y_p - o_p)^2$ (n is the size of y_p and o_p)

If the difference between two consecutive SSE is lower than a fixed tolerance or we have reached the maximum number of iteration stop and apply the inverse standardization of the output o, otherwise continue.

end function

Instead, the second algorithm, which computes v^* and v such as the previous one, gives the final output $o_p = b + \sum_{h=0}^{H} xh(h)v_p(h)\ \forall p$. Because the two algorithms were built with different purposes they have distinct input's domain: nnbackprop trains with 11 point while nnpredict generalizes the learning phase estimating the total average cost in 1000 points.

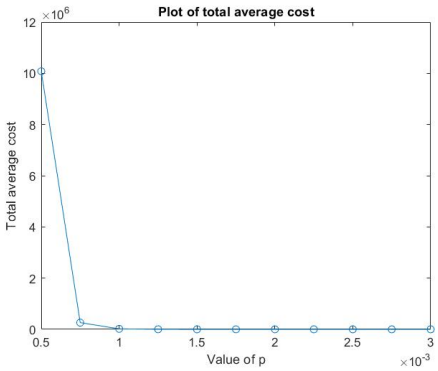
Analysis and Results

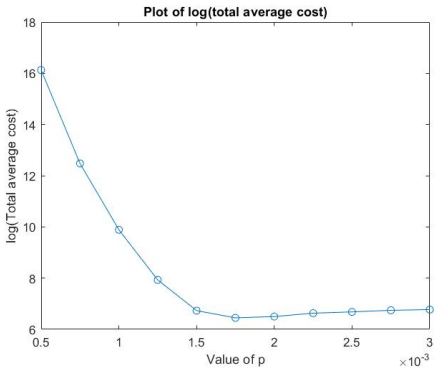
Observations about the code

At the beginning we built two matrices 60x1000, called In and Sn, which contain in each column a simulation of the Reed-Frost model over 60 days. The health cost of each simulation for a fixed p is obtained by the sum along the columns of the infected In. The total average cost is instead the mean of all these health cost plus the containment cost associated to the respective value of p. The final formula written in matlab is given by:

$$avcost(p) = mean(sum(In)) + (0.003/pgrid(p))^9 - 1;$$

In addition, we choose to work with the logarithm of the total average cost instead of the simple average cost. We thought this helped to perform a better training of the neural network because the logarithm reduces visual differences in datas with wide range. In fact the differences are proportional to percentages and this allowed us to include all the possible values of p. An alternative solution could be excluding the smallest values of p which determine the highest values of total average cost.





Another useful trick was to standardize the input and apply an inverse standardization of the output at the end. We did it both in nnbackprop and nnpredict because the sigmoid function takes in input values in [-2,2], otherwise it is very close to 0 or 1.

Best and Worst results

In order to find the best and the worst results we considered the number of input nodes (I), the tolerance (tol) and the maximum number of iterations (m_{max}) as fixed parameters. In particular, we assumed I=2, tol=$1e-7$, m_{max}=10000. Other solutions with different values could be possible as well.

For the number of hidden nodes (H) and the stepsize (mu) we applied a gridsearch. We considered all the odd H between 3 and 13 and 10 values of mu equally spaced between 0.01 and 0.1. To compute the optimal combination (H,mu), we need a quantity that measures the the approximation error of the neural network. Apart from considering the SSE we decided to compare graphically the results given by the trained neural network with the real initial simulations. To do this, we firstly predicted the total average cost over 1000 values of p in [0.0005,0.003] using nnpredict. Secondly, we linearly interpolate the couples (initial value of p, total average cost) in order to make possible the comparison with the results of nnpredict. Finally, the optimal combination of H and mu will be the one which minimizes the difference of the total average cost computed in the two distinct ways. All the things above are written in the following code:

```
% here we pick the maximum of the difference between 1000 points
% predicted by the neural network and 1000 points computed by the
% interpolation.
% i and j are the indeces of a particular combination (H,mu).
grid(i,j) = max(abs(ypred'-avcost_pred));
% now we take the best combination
mingrid=min(grid,[],'all','linear');
[Hsim(t),musim(t)]=find(grid==mingrid);
% and the worst one
maxgrid=max(grid,[],'all','linear');
[Hmax(t),mumax(t)]=find(grid==maxgrid);
```

To be more precise we repeted the entire process 100 times. This must be done in order to have strong results since the simulation of infected, susceptibles and the backpropagation's

parameters are randomly initialized. Here we report the frequency of the best and worst (H,mu) combinations, excluding the less relevant ones. Moreover we plotted the Neural Network prediction and the SSE only for the most frequent couples.

(H,mu)	frequency
(3,0.06)	20
(3,0.07)	17
(3,0.05)	17
(3,0.04)	15
(3,0.03)	10
(3,0.08)	7
(5,0.04)	4

Table 1. Best (H,mu) combinations

(H,mu)	frequency
(9,0.1)	22
(11,0.08)	17
(13,0.08)	17
(11,0.09)	14
(11,0.1)	7
(13,0.07)	7
(13,0.01)	3

Table 2. Worst (H,mu) combinations

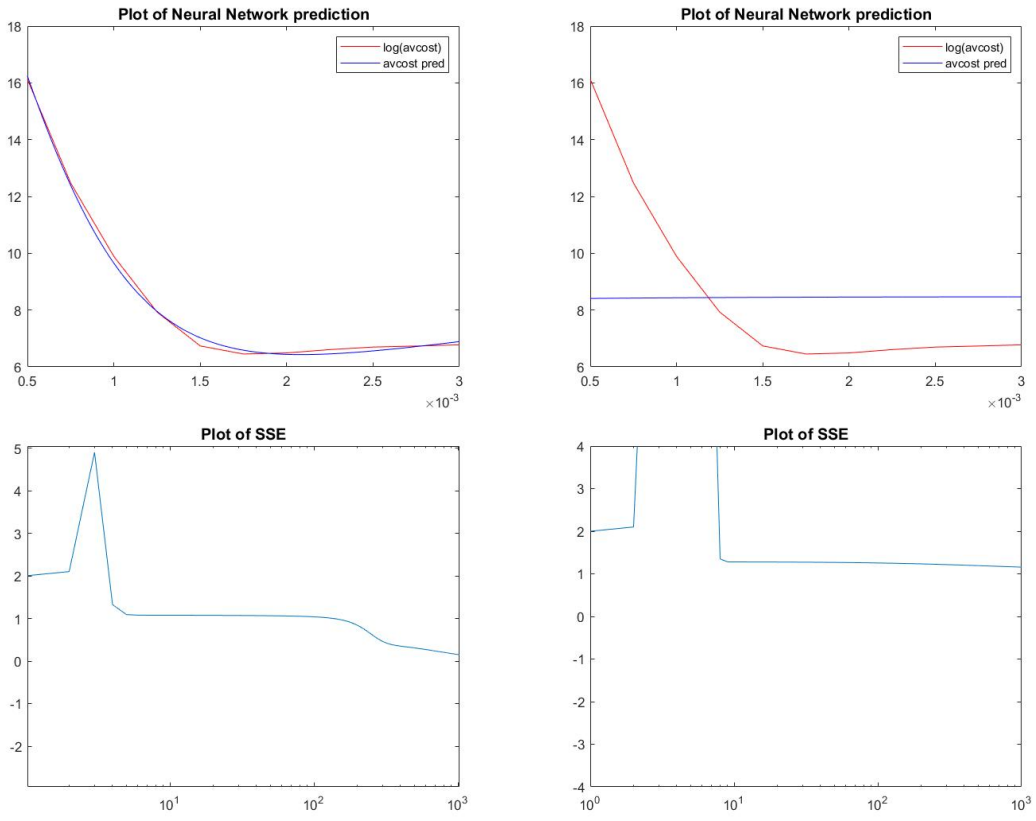


Figure 1. (H,mu)=(3,0.06) on the left and (H,mu)=9,0.1 on the right

In general a small H and a value of mu around 0.05 give better approximations than a high H and a mu near 0.1. This behaviour is coherent both in the plot of the neural network's predictions over 1000 points and in the plot of the SSE. In fact, as we can see in the last figure, the left SSE goes to zero along simulations. Moreover, there is a better correspondance between the logarithm of the total average cost (log(avcost)) and the predicted cost (avcost pred). On the contrary, the right figure (H=9, mu=0.1) is exaclty the opposite because SSE doesn't go to zero quickly and the neural network explodes. Finally, referring to the first plot on the left, we found numerically that the probability of one to one infection that minimizes the total average cost is p=0.002.

Appendix

```
% project1test script
% initialization
n=1000; % number of simulations
pgrid=[0.003, 0.00275, 0.0025, 0.00225, 0.002, 0.00175, 0.0015...
     0.00125, 0.001, 0.00075, 0.0005]; %probability of one to one ...
         infection

% inizialization of susceptible
Sn = zeros(60,1000);
S0 = 999;
Sn(1,:) = S0;

% inizialization of infected
In = zeros(60,1000);
I0 = 1;
In(1,:) = I0;

avcost = zeros(length(pgrid),1);
Hsim = zeros(20,1);
musim = zeros(20,1);
```

```
Hmax = zeros(20,1);
mumax = zeros(20,1);

for t = 1:100

    % simulations
    for s = 1:length(pgrid)
        for k = 1:1000
            for i=1:59
                In(i+1,k) = binornd(Sn(i,k),1-(1-pgrid(s))^(In(i,k)));
                Sn(i+1,k) = Sn(i,k)-In(i+1,k);
            end
        end
        avcost(s) = mean(sum(In))+(0.003/pgrid(s))^(9)-1;
    end

    % backpropagation + gridsearch for H and mu
    grid = zeros(6,10);
    i = 1;
    j = 1;
    for H = 3:2:13
        for mu = 0.01:0.01:0.1
            [o,xh,wih,SSE,b] = nnbackprop_vitt(pgrid,log(avcost),mu,H);
            [o,log(avcost)];
            xpred = linspace(0.0005,0.003,1000);
            ypred=interp1(pgrid,log(avcost),xpred,'linear');
            [avcost_pred] = ...
                nnpredict_vitt(wih,xh,xpred,H,log(avcost),b);
            grid(i,j) = max(abs(ypred'-avcost_pred));
            j = j+1;
        end
        j = 1;
        i = i+1;
    end

    mingrid = min(grid,[],'all','linear');
```

```
55      [Hsim(t),musim(t)] = find(grid==mingrid);
56      maxgrid = max(grid,[],'all','linear');
57      [Hmax(t),mumax(t)] = find(grid==maxgrid);
58  end
59
60  [Hsim, musim]
61  [Hmax, mumax]
62  % Looking at [Hsim, musim] and [Hmax, mumax] we found that
63  % H=3, mu=0.06 is the best combination while H=9, mu=0.1
64  % is the worst one.
65  % Here we report as a comment the functions we used to
66  % make the plot of the report:
67
68  % plot(xpred,ypred,'r',xpred,avcost_pred1,'b');
69  % title('Plot of Neural Network prediction','FontSize',12);
70  % legend('log(avcost)','avcost pred');
71
72  % plot(pgrid,avcost,'-o')
73  % xlabel('Value of p')
74  % ylabel('Total average cost')
75  % title('Plot of total average cost')
76
77  % semilogx((1:length(SSE)),(SSE))
78  % axis([0 1000 -4 4])
79  % title('Plot of SSE','FontSize',12);
```

```
1  function [o,xh,wih,SSE,b] = nnbackprop(xp,y_avcost,mu,H)
2
3      % step1: initialization
4      I = 2; % number of input nodes
5      m_max = 10000; % max number of iterations
6      n = length(xp);
7      tol = 10^(-7);
8
```

```
9       % error sum of square
10      SSE = 2*ones(m_max,1);
11      SSE(2) = 2.1;
12
13      wih = 0.5*rand(I,H);
14      xh = 0.5*rand(H,1);
15      b = unifrnd(0,0.5);
16
17      xp_norm = (xp-min(xp))/(max(xp)-min(xp)); % normalized input
18      up = [ones(n,1) xp_norm'];
19
20      vstar = ones(n,H);
21      v = ones(n,H);
22      o = ones(n,1);
23
24      y_avcost_norm = ...
            (y_avcost-min(y_avcost))/(max(y_avcost)-min(y_avcost)); % ...
            normalized y_avcost
25
26      m = 0;
27      k = 2;
28      while abs(SSE(k)-SSE(k-1)) > tol && m ≤ m_max
29
30          % step2-3: computation of v* and v
31          for h = 1:H
32              for p = 1:n
33                  vstar(p,h) = sum(wih(:,h).*(up(p,:))');
34                  v(p,h) = 1/(1+exp(-vstar(p,h)));
35              end
36          end
37
38          % step4: computation of the output o
39          for p = 1:n
40              o(p) = b+sum(xh.*(v(p,:)'));
41          end
42
```

```
43          % step5: update of wih and xh
44          b = b+mu*sum(y_avcost_norm-o);
45          for i = 1:I
46              for h = 1:H
47                  wih(i,h) = wih(i,h)+mu*sum((y_avcost_norm-o)*...
48                      xh(h).*v(:,h).*(1-v(:,h)).*up(:,i));
49              end
50          end
51          for h = 1:H
52              xh(h) = xh(h)+mu*sum((y_avcost_norm-o).*v(:,h));
53          end
54
55          % step6: computation of SSE
56          m = m+1;
57          k = k+1;
58          SSE(k) = sum((y_avcost_norm-o).^2);
59      end
60
61          % step7: inverse normalization of the output
62          o = (max(y_avcost)-min(y_avcost))*o+min(y_avcost);
63  end
```

```
1   function [avcost_pred] = nnpredict(wih,xh,xpred,H,y_avcost,b)
2
3       % inizialization
4       n = length(xpred);
5       xpred = (xpred-min(xpred))/(max(xpred)-min(xpred));
6       up = [ones(n,1) xpred'];
7       vstar = ones(n,H);
8       v = ones(n,H);
9       o = ones(length(xpred),1);
10
11      % computation of vstar and v
12      for h = 1:H
```

```
13          for p = 1:length(xpred)
14              vstar(p,h) = sum(wih(:,h).*(up(p,:))');
15              v(p,h) = 1/(1+exp(-vstar(p,h)));
16          end
17      end
18
19      % computation of the output o
20      for p = 1:length(xpred)
21          o(p) = b+sum(xh.*(v(p,:)'));
22      end
23
24      % inverse normalization of the output
25      avcost_pred = (max(y_avcost)-min(y_avcost))*o+min(y_avcost);
26  end
```