Stochastic Optimization - Project 1

Cagnasso Pietro
Student id: s300801
s300801@studenti.polito.it

Vergaro Nicolò
Student id: s295633
s295633@studenti.polito.it

I. Problem overview

The project we worked on is the Reed-Frost model for epidemic spread. It divides the population into:

- Susceptible
- Infected
- Removed

The system's state is entirely described by the pair (s, i), since r, given a population of size N, is always described as:

$$r = N - s - i \tag{1}$$

The unit of time in our project is a day. This model sets the infection period to a unit of time, removing a person infected at time t at time $t+1$. Each infected person at time t can infect each susceptible person with a probability p. We simulated the system as follows:

$$(S_n, I_n) \to \begin{cases} I_{n+1} \sim Bin(S_n, 1-(1-p)^{I_n}) \\ S_{n+1} = S_n - I_{n+1} \end{cases} \tag{2}$$

Each infected person will surely recover, but at the cost $c_{health} = 1$ for the health system. The probability of infection is $p = 0.003$ without any intervention. The government can apply measures to reduce the likelihood of infection, but this has an impact on the economy. For $p \in [0.0005, 0.003]$ the cost on economy is defined as:

$$c_{economy}(p) = (\frac{0.003}{p})^9 - 1 \tag{3}$$

II. Proposed approach

We addressed the problem by dividing it into three sub-problems:

- Simulation
- Function fitting with neural network
- Finding p corresponding to the minimum

A. Simulation

We simulated a system with $N_{people} = 1000$ people, starting with the initial state $(S_0, I_0) = (999, 1)$.

We simulated the process $N_{sim} = 1000$ times for $N_{days} = 60$ consecutive days with infection probabilities ranging from 0.0005 to 0.003, with a step equal to 0.00025.

Fig. 1 shows the epidemic trend in the population for different infection probabilities. We can see that we got a flat trend for $p = 0.0005$. The use of a binomial random variable explains this behavior: since we compute $I_1 \sim Bin(S_0, 1-(1-p)^{I_0})$, whose expected value is $S_0(1-(1-p)^{I_0}) = 0.4995$, thus *on average* no more infections will be encountered starting from the second day.

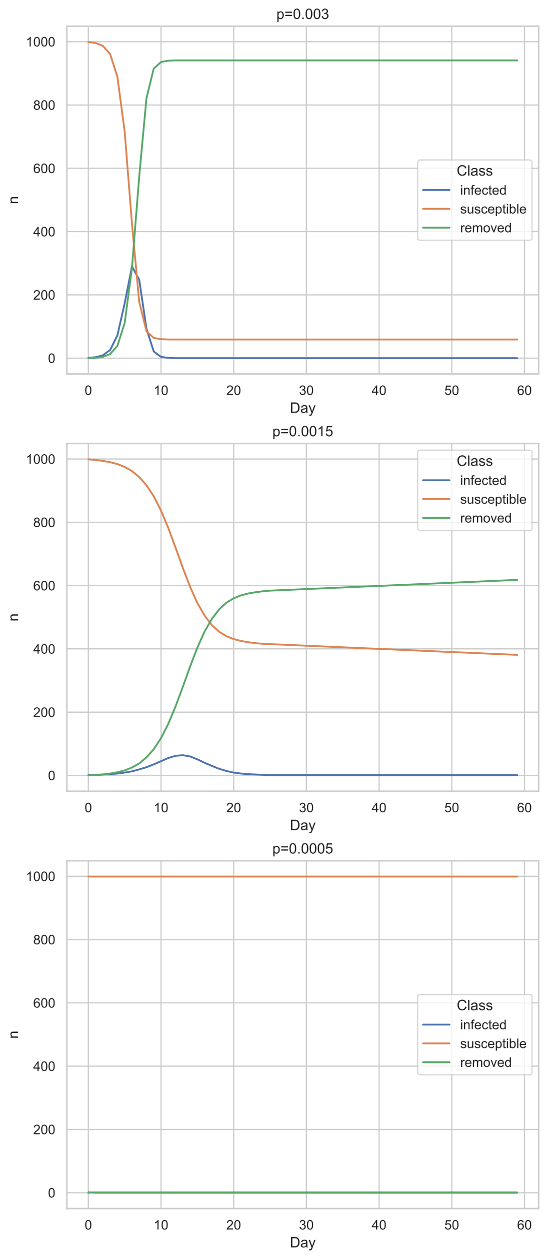

Fig. 1. Infection trend in the population with $p = 0.003$ (top), $p = 0.0015$ (middle), $p = 0.0005$ (bottom).

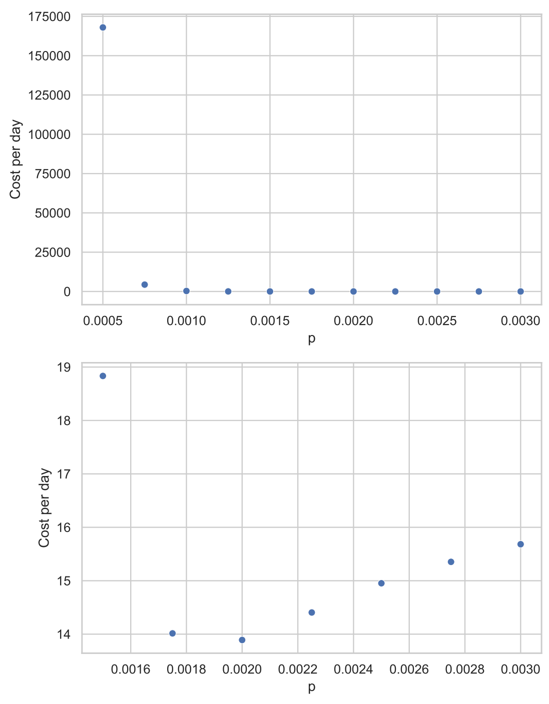

Fig. 2. Average cost per day based on the p of choice. The bottom figure shows a zoom on the more interesting portion of the chart.

Using the information from the infection curve, we can compute the average cost per day. It is computed as:

$$c_{daily}(p) = \frac{c_{economy}(p) + c_{health} \cdot TOT_{infected}}{N_{days}} \tag{4}$$

Fig. 2 shows the average cost per day that comes from the simulations. We can see that the cost tends to explode for lower probability values: this is its behavior even if we computed that, on average, no more infections would be found after the first day. In the same figure in the bottom image, we can appreciate the clear presence of a minimum.

B. Function fitting

From the simulation we got a point-wise estimation of the value of the cost function. To find the minimum, we needed a more refined estimate of our process, so we used a neural network to fit our points.

As a good rule of thumb, when using neural networks, we scaled the inputs and outputs. Our ranges of choice were:

- $x \in [-2, 2]$ so that the activation function can use almost its entire variation
- $y \in [0, 1]$ since the output of the activation function we chose is also in the same interval, so avoid the explosion of weights

The sigmoid activation function we used for the hidden layer is modified to be able to fill almost its entire variation range:

$$\sigma(u) = \frac{1}{1 + e^{-4u}} \tag{5}$$

In fact with this change the range covered is $[0.0003, 0.9997]$ instead of $[0.1192, 0.8808]$ without considering the weights.

TABLE I
SUMMARY OF HYPERPARAMETERS USED IN THE IMPLEMENTED NEURAL NETWORK.

Hyperparameter	Value
hidden nodes	5
bias bound	0.5
weight bound	0.3
tol	10^{-8}
learning rate	Eq. 6
max epochs	10^6
standardization input range	$[-2, 2]$
standardization output range	$[0, 1]$

In the output layer, we used a linear activation function that is an identity, so it has no range limitations.

We initialized the input and hidden layer weights with a uniform random variable between 0 and 0.5. We used the same approach to initialize the input and output bias using a uniform random variable between 0 and 0.3. We set the tolerance parameter to 10^{-8} and the learning rate is computed as:

$$\mu(t) = 10^{-4}\sqrt{t} \tag{6}$$

where t is the epoch. We set the maximum number of epochs to 10^6.

The layout we chose for our neural network has five neurons in the hidden layer. Both the input layer and the hidden layer have an additional bias neuron. Fig. 3 shows the general structure of the implemented neural network.

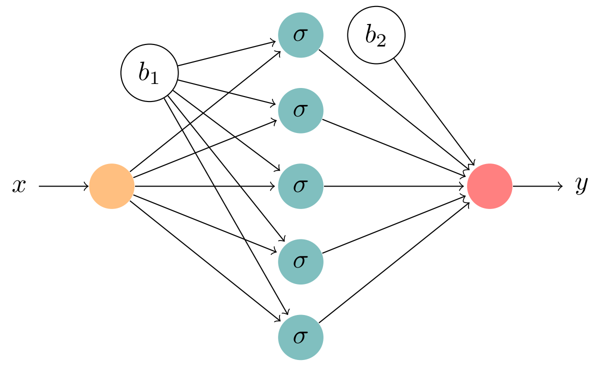

Fig. 3. Implemented neural network general layout.

TABLE I is a summary table of the hyperparameters used for the neural network.

We decided to implement the algorithm through a matrix-based approach, to reduce the computation time.

1) Forward propagation: We called $X \in \mathbb{R}^{(r \times i)}$ (where i is the number of inputs) the column vector containing the values of infection probability, $y \in \mathbb{R}^{(r \times o)}$ (where o is the number of outputs) the column vector given by the simulation at each infection probability. To add the bias input we horizontally stacked a column vector of ones to X:

$$\hat{X} = [X | 1_{(r \times 1)}] \Rightarrow \hat{X} \in \mathbb{R}^{(r \times i+1)} \tag{7}$$

We computed the matrix of weights for the inputs by extracting them from a $Uniform(0, wieght\_bound)$ obtaining

$W \in \mathbb{R}^{(i\times h)}$ (where h is the number of hidden nodes) and the actual values of the input bias from $Uniform(0, bias\_bound)$ obtaining $B_i \in \mathbb{R}^{(1\times h)}$. Then we vertically stacked those two row vectors into:

$$\hat{W} = [W^T | B_i^T]^T \Rightarrow \hat{W} \in \mathbb{R}^{(i+1\times h)} \tag{8}$$

We computed the inputs for the hidden layer as:

$$V^* = \hat{X} \times \hat{W} \Rightarrow V^* \in \mathbb{R}^{(r\times h)} \tag{9}$$

using this intermediate result we applied the activation function element-wise obtaining V. Once again to add the bias we horizontally stacked a columns vector of ones to V:

$$\hat{V} = [V | 1_{(r\times 1)}] \Rightarrow V \in \mathbb{R}^{(r\times h+1)} \tag{10}$$

We computed the matrix of weights for the outputs of the hidden layer by extracting them from $Uniform(0, wieght\_bound)$ obtaining $K \in \mathbb{R}^{(h\times o)}$ and the actual values of the input bias from $Uniform(0, bias\_bound)$ obtaining $B_h \in \mathbb{R}^{(1\times o)}$. Then we vertically stacked those two row vectors into:

$$\hat{K} = [K^T | B_h^T]^T \Rightarrow \hat{K} \in \mathbb{R}^{(h+1\times o)} \tag{11}$$

We computed the inputs to the output layer as:

$$O = \hat{V} \times \hat{K} \Rightarrow O \in \mathbb{R}^{(r\times o)} \tag{12}$$

since the output layer has an identity activation function we have O as the output.

2) *Back propagation:* We computed the error as $e = y - O$ and used it to compute how to change the weights.

We computed how much we should change the input weights as:

$$\Delta W = \hat{X}^T \times (((e \cdot deactivate(O)) \times \hat{K}^T) \cdot deactivate(\hat{V})) \\ \Rightarrow \Delta W \in \mathbb{R}^{(i+1\times h+1)} \tag{13}$$

To update those weights we used only *up to the second last column* (ΔW^*) of the computed matrix because the last one refers to the hidden bias:

$$\hat{W} = \hat{W} + 2 \cdot lr(t) \cdot \Delta W^* \tag{14}$$

We computed how much we should change the hidden weights as:

$$\Delta K = \hat{V}^T \times (e \cdot deactivate(O)) \Rightarrow dK \in \mathbb{R}^{(h+1\times o)} \tag{15}$$

and updated the weights as:

$$\hat{K} = \hat{K} + 2 \cdot lr(t) \cdot \Delta K \tag{16}$$

We processed with the trained neural network an array of 10001 linearly spaced elements in $[0.00125, 0.003]$ and we obtained the results shown in Fig. 4.

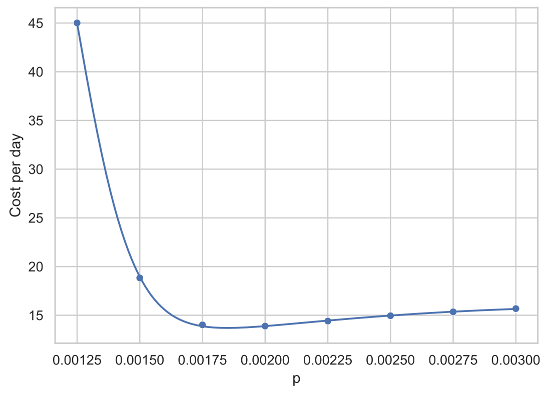

Fig. 4. Output of the neural network with highlighted the point used to train the network.

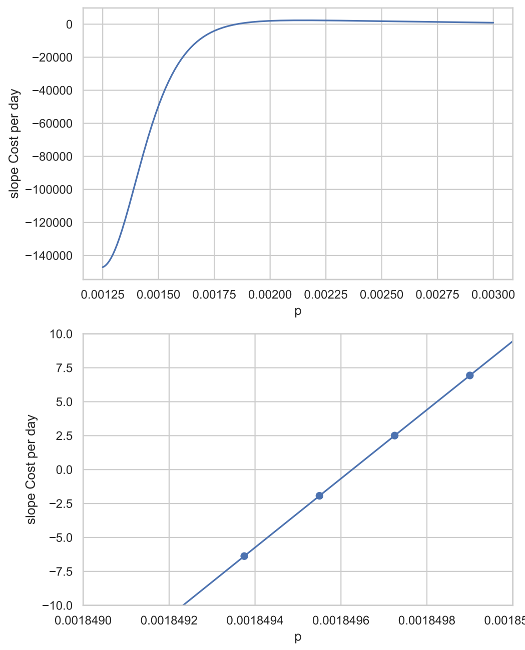

Fig. 5. Slope of the function obtained form the neural network, in the bottom figure there is a zoom in the minimum zone.

C. Find the minimum

Looking at the output of the neural network we know that to find the minimum we have to search for the first change in sign of the derivative. Since the function we have is not continuous we can only work on the slope between each pair of points, computed as:

$$slope(i) = \frac{c_{i+1} - c_i}{p_{i+1} - p_i} \tag{17}$$

From the resulting function we can retrieve the point of minimum. From Fig. 5 we can set the minimum for the output of the neural network as the value obtained for $p_{min} = 0.001849725$. If we approximate the derivative as a line in the interval $[0.00184955, 0.001849725]$, we can compute the p of the minimum obtaining: $p_{min} = 0.00184963$.

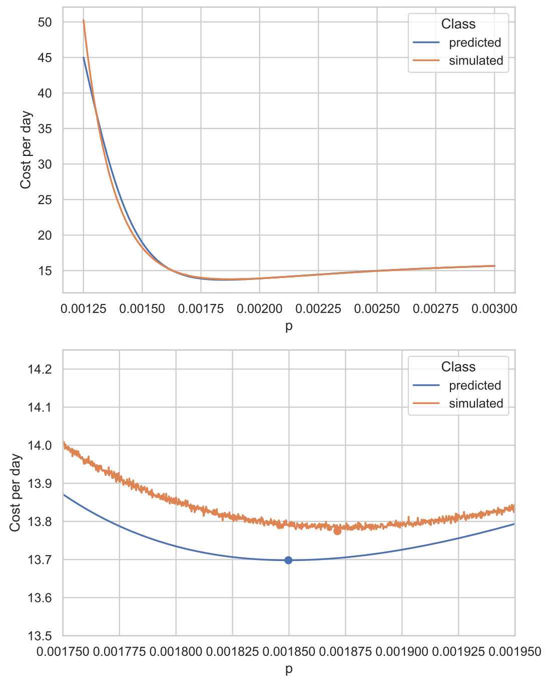

Fig. 6. Comparison of the simulation and its approximation, in the bottom the highlighted points are the minima.

III. Conclusion

Since the problem we worked on is quite simple we had the opportunity to make the simulation of the system in each point in which we computed the approximation. To make the result of the simulation less noisy we increased the number of simulations up to 10000.

The minimum find by the simulation is

$$min_{sim} = (0.001871425, 13.774568733)$$

while the neural network approximation has its minimum in

$$min_{nn} = (0.001849725, 13.698125361)$$

and finally the minimum computed by the neural network in the p_{min} we computed approximating the derivative with the slope is in

$$min_{slope} = (0.00184963, 13.13.69812546)$$

Those results are similar, in fact the function approximation obtained from the neural network is quite good resulting in:

- $MSE = 0.418$
- $MAE = 0.255$

computed on the entire range $[0.00125, 0.003]$ Fig. 6 shows the comparison between the function approximation and the more precise simulation we computed.