Machine Learning Project Report : Simulated Annealing

Bharath Rabindranath

January 6, 2017

Contents

1 Introduction **2**

2 The Task at Hand **2**

3 The Functions in the Code **4**
3.1 newmicrostate(current_ms, height, width) 4
3.2 energyCalculationdiff2(microstate, width, selected_row2, selected_column2,selected_row1, selected_column1) 4
3.3 Simulated Annealing 4

4 Results and Calculations **6**
4.1 A first result 6
4.2 The Second Result 6
4.3 The Third Result 7
4.4 The Final Result 8

5 Improvements **9**

6 Acknowledgements **10**

1 Introduction

Simulated Annealing (SA) is a general purpose probabilistic technique for approximating a global optimum of a given arbitrary complicated function in a large state space. Very often this method is used when we want to find an approximation to a global optimum and not an exact solution of a local optimum. SA is often used when the search space is very large and discrete. For example, Prof. Herbert Jaeger's notes summarize how the algorithm may be used to find a approximate solution to the Traveling Salesman Problem. Here, one can imagine the state space to be the set of all paths between a given set of cities.

The key idea behind this algorithm comes from Metallurgy. Very often, metals are heated and the cooled in a controlled fashion allowing for enlargement of it's crystals and a reduction in it's defects. The metal is cooled down to a point where it is said to be "frozen". The point at which it is said to be "frozen" happens at a minimum energy configuration.

2 The Task at Hand

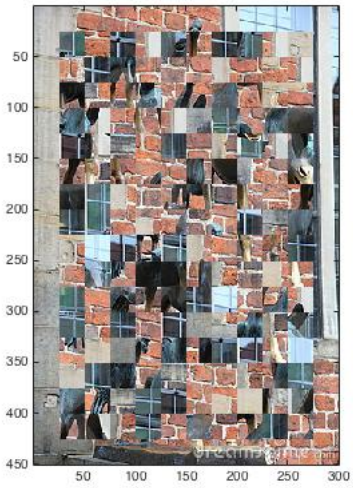
In this project, given a scrambled image in a MATLAB 4d array format $L * L * 3 * N$. Where $L = 25$ is the tile length, $N = height \cdot width$ is the product of the height and width of the image and the 3 corresponds to the color channels (R, G, B).

The task is to unscramble the image using Simulated Annealing as the main underlying routine. A picture of the original unscrambled image is given below

To let the Simulated Annealing algorithm work it's magic there are three key aspects that need to be defined.

- State Space : I have defined my state space as the set of all images that satisfy the dimensions of the original unscrambled image that is $L * L * 3 * N$. This includes the unscrambled image and all other possible scrambled images.
- Method to Explore the State Space: There were two approaches that I thought possible for this
  - Moving from one scrambled image to another scrambled image picked at random. This would involve computing a "total energy" for an entirely new configuration.
  - Randomly picking two tiles in the image and swapping them. This does not involve computing a "total energy" for the whole

Figure 1: Original Unscrambled Image



picture but instead requires a total energy calculation only for the two swapped tiles. The energy contributed by all the other tiles remains the same and when calculating the difference cancels out.

- An Energy Function: The Energy Function assigns a microstate (ms) which is an element of our State space an Energy. Once, a measure of the energy can be calculated for two states we can take the difference of the total energy to get δE.

3 The Functions in the Code

3.1 newmicrostate(current_ms, height, width)

The function $newmicrostate(current\_ms, height, width)$ does exactly what is described in the second sub-item of the second bullet point mentioned above. The function takes as input a current microstate($current\_ms$), the $height$ and the width of the image which remain constant throughout. The output of the function is a new microstate(new_ms), the indices i.e the row and column of the first tile that is swapped ($selected\_row1, selected\_column1$) and the indices i.e the row and column of the second tile that is swapped ($selected\_row2, selected\_column2$).

3.2 energyCalculationdiff2(microstate, width, selected_row2, selected_column2,selected_row1, selected_column1)

$energyCalculationdiff2(microstate, width, selected\_row2, selected\_column2, selected\_row1, selected\_column1)$ assigns an energy to two microstates namely the state prior to the swaps of two randomly selected tiles and the state after the swaps of the two randomly selected tiles. It returns the difference in energy δ_e. The energy is calculated by looking at the neighboring immediate pixel vectors of both the selected tiles to be swapped prior to the swap and post the swap. More formally, the energy given two pixel vectors $x_i^{(j)}$ and $y_i^{(j)}$ where $1 \leq i \leq P$ is an index running over the length of the pixel vectors and $1 \leq j \leq 3$ is an index running over the color channels is defined as

$$\sqrt{\frac{1}{3} \cdot \sum_{j=1}^{3} \frac{1}{P} \cdot \sum_{i=1}^{P} (x_i^{(j)} - y_i^{(j)})^2}$$

We compare the pixel vectors immediately above, immediately left, immediately right and immediately below the two selected tiles prior to swapping them and post swapping them. That is a total of $4 * 2 * 2 = 16$ operations. The rest of the tiles as mentioned above have not changed in energy, so when calculating the difference add no effect to δ_e.

3.3 Simulated Annealing

The last file that acts a wrapper around all the procedures is "SimulatedAnnealing.m". This file is the main heart of the program. The structure of the file is the basic idea of the algorithm. The procedure is given in pseudocode below

$file\_name \leftarrow shuffledImageEasy.mat$
$current\_ms \leftarrow RGBtilesShuffled$
$temp \leftarrow 150$
$temp\_minimum \leftarrow 0.5$
$zeta \leftarrow 0.99$
while $temp > temp\_minimum$ **do**
    $accepted\_better \leftarrow 0$
    $accepted\_worse \leftarrow 0$
    $rejected \leftarrow 0$
    for $1 \leq i \leq 10700$ **do**
        $[new\_ms, row1, column1, row2, column2] \leftarrow$
$newmicrostate(current\_ms, height, width)$
        $delta\_e \leftarrow$
$energyCalculationdiff2(new\_ms, width, row2, column2, row1, column1)$
        if $0 \leq delta\_e$ **then**
            $current\_ms \leftarrow new\_ms$
            $accepted\_better \leftarrow accepted\_better + 1$
        else
            $acceptance\_probability \leftarrow exp(\frac{delta\_e}{temp})$
            if $rand() \leq acceptance\_probability$ **then**
                $current\_ms \leftarrow new\_ms$
                $accepted\_worse \leftarrow accepted\_worse + 1$
            else
                $rejected \leftarrow rejected + 1$
            end if
        end if
    end for
    $temp \leftarrow zeta \cdot temp$
printing $temp$
end while

4 Results and Calculations

We may use the following formula to find out the total number of iterations for the whole program

$$total\_iterations = \frac{\log\left(\frac{MinimumTemp}{StartingTemp}\right)}{\log\left(zeta\right)} \cdot iter\_per\_temp$$

Below are the following results I got after trying out the program with various parameters

4.1 A first result

Figure 2: First Result Achieved

The above picture was the first result achieved. This was got by setting the $StartingTemp$ to 150, the $MinimumTemp$ to 0.25, $zeta$ to 0.973 and $iter\_per\_temp$ to 12900. Approximately that is $3.01 * 10^6$ iterations. We can see that the glass windows and the bricks have started to get constructed. The others parts are clustered correctly but are in the wrong places. Perhaps, I cooled to fast was my intuition here. If I were to grade this using the standard Jacob's University grading pattern with 1.0 being perfectly unscrambled and 5.0 meaning no change. This result would be a 3.3.

4.2 The Second Result

The picture below was the second result achieved. This was got by setting the $StartingTemp$ to 150, the $MinimumTemp$ to 0.25, $zeta$ to 0.975 and

Figure 3: Second Result Achieved

iter_per_temp to 15000. Approximately that is $3.78 * 10^6$ iterations. Here we see that the glass windows are constructed almost completely. More than the previous result. We see the bricks are also almost constructed completely. We also see the clump of the glass window tiles at the base of the statue indicating maybe that whilst they are matched up correctly, I might have cooled the system a bit too slow. Grading this on the Jacob's University grading pattern, the result would be a 3.0.

4.3 The Third Result

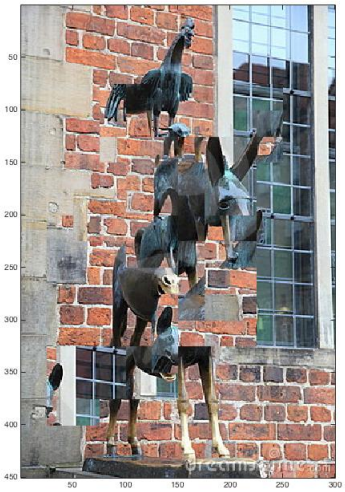
The picture below was the third result achieved. This was got by setting the $StartingTemp$ to 210, the $MinimumTemp$ to 4.9, *zeta* to 0.99 and *iter_per_temp* to 10310.Approximately that is $3.85 * 10^6$ iterations. The number of iterations in total are in fact more even though the iterations per temperature is around 4000 less than before. The parameters were set to these after thinking about how a metal cool and trying to mimic the continuous cooling of a hot piece of metal. This is why the *zeta* value has been changed to 0.99 making the jumps between the temperatures a lot smaller. We see that the glass and the bricks are completely reconstructed but the animals are not. My idea was to maybe let it hold a little longer on a single temperature value. That is increase the iterations per temperature value. Grading this on the Jacob's University grading pattern, the result would be a 2.6.

Figure 4: Third Result Achieved

4.4 The Final Result

The picture below was the fourth result achieved. This was got by setting the $StartingTemp$ to 210, the $MinimumTemp$ to 4.9, $zeta$ to 0.99 and $iter\_per\_temp$ to 11000.Approximately that is $4.11 * 10^6$ iterations. The number of iterations in total are in fact more even though the iterations per temperature is around 4000 less than the second result. . Once, again this parameters were set with the intent of mimicking how a metal would cool in a continuous manner although I am holding it at a temperature a while longer by running it for 690 iterations per temperature more than in result 3. This is the best result achieved. We see that the glass is almost completely reconstructed barring the small cluster that is located towards the hind legs of the donkey. The animals have lined up now one on top of the other. The rooster is almost completely reconstructed and so are the legs of the donkey at the base of the statue. Grading this on the Jacob's University grading pattern, the result would be between 2.33 and 2.67.

Figure 5: Final Result Achieved



5 Improvements

A lot of improvements can be made to the energy function in the algorithm. For example instead of just comparing the neighboring pixel vectors, we can compare the neighboring three pixel rows in all directions and computer a gradient of color change. This might give us better results. Another idea might be to keep track of the free energy of the system.

Perhaps instead of swapping two random tiles to get to the next microstate, selected another random scrambled image as a whole might lead to quicker results. This might be due to the bigger jumps in energy. This might happen if for example we move from a configuration that is relatively scrambled suddenly (out of luck) to a relatively unscrambled image.

Another idea might be to incorporate global and local components into the energy function. Incorporating these ideas might make for a better model as a whole. Unfortunately, this improvements could not be attempted in the duration given.

6 Acknowledgements

I would like to extend my gratitude and thanks to Prof.Dr. Herbert Jaeger for giving me the opportunity to work on this extremely interesting project and for providing me with the raw scrambled image.