Submodularity: approximating the optimal solution of a combinatorial problem

Jorik Jooken, Michiel Baptist

KULeuven

March 2019

Structure

The structure of this report is as follows: first we give a very brief motivation of why the topic of submodularity is interesting. We then list the most important sources that we used to come to the idea for the creative part. We make the assumption that the reader is already familiar with submodularity and hence we have chosen not to repeat any theory or definitions. The next part forms the core of the report and focuses on the creative part that we have worked out for this course. After this section, the results are presented. Finally, the report is concluded with our main observations.

Motivation

Many problems in machine learning require an optimisation problem to be solved. Some examples of objective functions that arise often in machine learning are: log-likelihood, information gain, sum of squared residuals... It is often a challenging task to devise an algorithm that is able to efficiently compute (an approximation of) a solution for the optimisation problem at hand. Hence, it is important that one tries to use the specific structure of the objective function in order to be able to rely on well-known algorithms for finding a solution. For convex optimisation problems for example, one can efficiently find a set of parameters that minimises the objective function by applying a gradient descent method [5].

In optimisation problems, one can make the distinction between continuous and discrete optimisation problems. In continuous optimisation problems, the parameters can take on values from a continuous range of values (for example the real numbers). In discrete optimisation problems on the other hand, one has to assign values from a discrete set of values to the parameters. Convexity is a property of continuous functions. There is a somewhat similar property that discrete functions can have: submodularity. The main topic on which this report focuses is a creative application in which a submodular function is optimised.

Sources

The most important sources that we used to get acquainted with the topic of submodularity are listed here. We started by skimming through the survey paper "Learning with Submodular Functions: A Convex optimisation Perspective" [1], in order to get an overview of the topic. Next, we followed the tutorial "Beyond Convexity: Submodularity in Machine Learning" [4] by A. Krause and C. Guestrin. Finally, we read the paper "SFO: A Toolbox for Submodular Function Optimisation" [3]. In this paper, a MATLAB toolbox for submodular function optimisation is described. This toolbox was the starting point for the creative part of the task. The part of the code in our algorithm that concerns the optimisation of submodular functions was not written by ourselves, but comes from this toolbox. The complete code can be found on GitHub [2].

The idea for the creative part mainly originated from the tutorial "Beyond Convexity: Submodularity in Machine Learning" [4]. In this tutorial, many applications regarding images are described. Furthermore, the tutorial contains an example of a graph theory problem (the maximum cut problem) in which a submodular function has to be optimised. The idea for the creative part tries combine these ideas: it tries to use the solution for the maximum cut problem in the context of images.

Creative part

For the creative part of the task, we have written a program that attempts to find structures in images. The goal of the program is to identify a set of pixels that represent (the contours of) one or more objects in an image. This set of pixels tends to be characterised by the fact that they are quite different from surrounding pixels. This insight forms the motivation for the method that we have implemented to solve this problem. A detailed overview of this method follows next.

Compressing the image

A typical image consists of several hundreds of thousands of pixels. Of all these pixels, we want to select a subset of pixels that represent (the contours of) one or more objects in an image. This subset of pixels will be determined by selecting a subset of variables in a particular submodular optimisation problem, that will be explained in the following subsection. The image in which we want to detect structures, is first compressed. This is done in order to keep the running time, needed to solve the optimisation problem, low.

The compression method that we used, is quite simple. Suppose that the original image consist of r rows of pixels, each containing c pixels. We denote an image as $\{x_{i,j}\}$ for $i \leq r$ and $j \leq c$. Here a pixel $x_{i,j}$ is a vector in $\mathbb{R}^3$ that denotes its RGB colour code. The image is first divided into blocks of size s by s. The parameter s is chosen for every image individually in such a way that the whole algorithm can be executed in several minutes. The compressed image will consist of $\frac{r}{s}$ rows, each containing $\frac{c}{s}$ pixels. The colour of a pixel $p_{i,j}$ in the compressed image is calculated as

$$p_{i,j} = \sum_{k=i.s}^{(i+1).s-1} \sum_{l=j.s}^{(j+1)s-1} \frac{x_{k,l}}{s^2} \tag{1}$$

(i.e. the average of all the colours of the corresponding block in the original image).

An example of this compression is given in figures 1 and 2. Figure 1 shows the original image and figure 2 shows the compressed image. Here, the block size $s = 8$ was used.

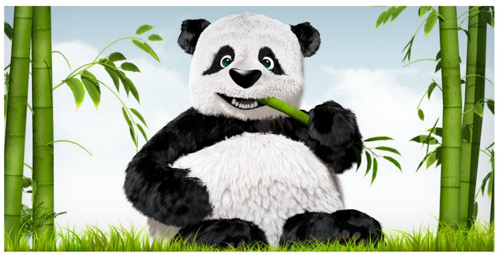

Figure 1: Original image

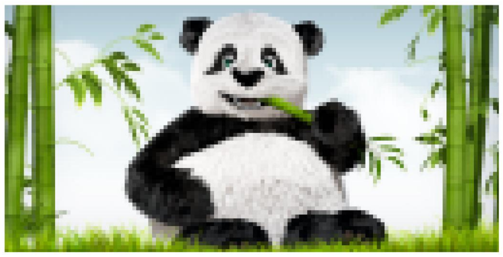

Figure 2: Compressed image

From image to graph

The compressed image is transformed into a weighted graph as follows. We construct a graph $G = (V, E)$ such that there is a node for every pixel and there is an edge between two nodes if and only if the corresponding pixels are direct neighbors (i.e. most pixels have eight neighboring pixels). The weight of an edge is a function of the dissimilarity between the two corresponding pixels. A pixel is represented by a vector in $\mathbb{R}^3$ and the dissimilarity between two pixels is defined as the Euclidean distance between these two vectors. Let e be an edge of the graph G and let p_1 and p_2 be the pixels corresponding to the nodes at the endpoints of this edge. Then, the weight of an edge is defined as follows:

$$w(e) = \begin{cases} 0, & \text{if } \|p_1 - p_2\| < \alpha \\ \|p_1 - p_2\|, & \text{otherwise} \end{cases} \tag{2}$$

Here, α is a parameter that indicates a treshold. The weight of an edge between neighboring nodes whose pixels have a dissimilarity below this treshold is equal to 0.

For this graph G, we want to find a set of nodes such that the sum of the weights of edges from nodes inside the set to nodes outside the set is large. These nodes will correspond to the pixels that represent (contours of) one or more objects in the image. This problem is known as the problem of finding a maximum weighted cut of a graph. The objective function (the cut function) is defined as follows:

$$F(A) = \sum_{v \in A} \sum_{u \in N(v) \setminus A} w_{v,u} \tag{3}$$

Here, $N(v)$ denotes the set of neighbors of the node v. This is a submodular function (proof in appendix) and hence the algorithms for submodular function maximisation can indeed be applied.

By solving this optimisation problem, a set of nodes is obtained. These nodes correspond to pixels in the compressed image. The blocks of the original image that correspond to these pixels are the pixels that will be marked as belonging to (the contour of) an object.

Results

In this section, we will show the output of the algorithm on some example figures and report on the parameters that were used to obtain these results.

In figure 3, we see the output of the algorithm when the input image is the image from figure 1. We can clearly see that the algorithm is able to detect several structures: the grass, the leaves, the ears, eyes and nose of the panda... The algorithm seems to misidentify two tiny spots (at the belly of the panda) that don't really represent any objects. Furthermore, the algorithm does not detect the existence of the clouds in the background. The original image of the panda has size 510 pixels by 1020 pixels. The block size s that we have chosen for this image is equal to 8. The parameter α is equal to 0.30.

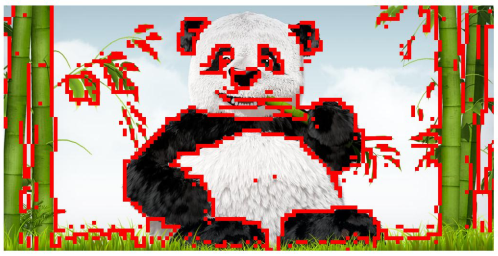

Figure 3: The result of running our algorithm on the example in figure 1

Another image for which we have tested the algorithm is shown in figure 4. The output of our algorithm can be found in figure 5. The algorithm is again able to detect several structures: the mouth, nose, eyeball, brain. . . It incorrectly marks several random pixels in front of the eyeball as part of (the contour of) an object. This might be related to the fact that the original image is compressed (such that the pixel in front of the eyeball in the compressed image is not entirely black and hence seen as part of the object). The original image of Homer Simpson has size 471 by 480 pixels. The block size s that we have chosen for this image is equal to 4. The parameter α is again equal to 0.30.

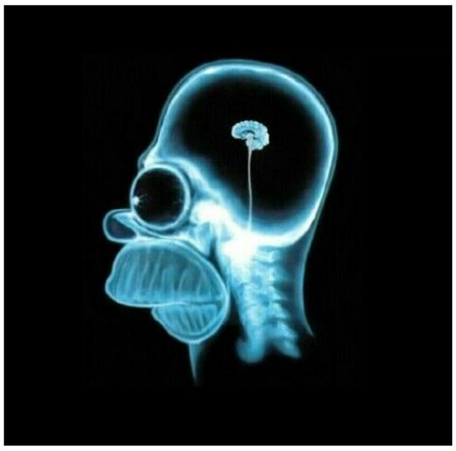

Figure 4: Original image

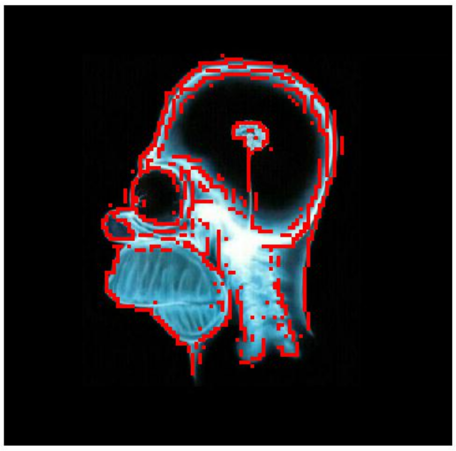

Figure 5: The result of running our algorithm on the example in figure 4

The parameter α can be varied to have an impact on the granularity of the detected structures in the image. Lower values of α correspond to more fine-grained structures being detected. This effect can be seen in figures 6, 7 and 8. Figure 6 contains the original image, which has size 873 by 973 pixels. The value $\alpha = 0.3$ was used for figure 7 and $\alpha = 0.5$ for figure 8. The block size s that we have chosen for both images is equal to 10.

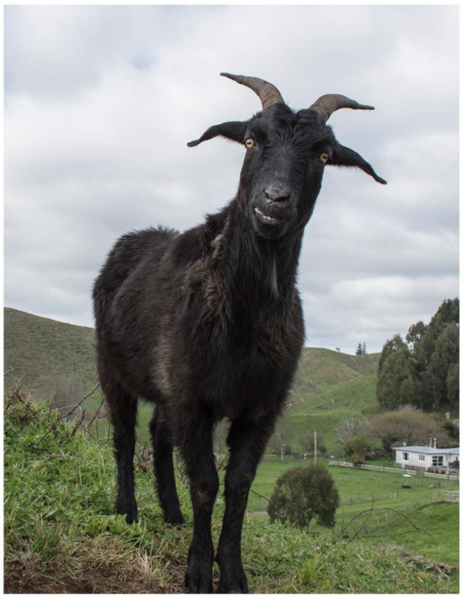
Figure 6: Original image

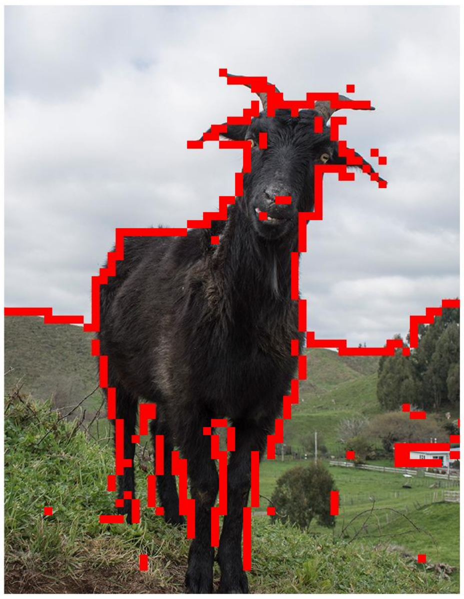
Figure 7: The result of running our algorithm on the example in figure 6 with $\alpha = 0.3$.

Conclusion

In this report we discussed the role of discrete optimisation problems in machine learning. We focussed more specifically on the topic of submodularity: the discrete counterpart to convexity in continuous optimisation. Furthermore, we elaborated on a creative application of optimising submodular functions, which consisted of finding structure in images. The problem of finding structure in images was translated to the graph theoretical problem of finding a maximum cut in a weighted graph. The examples in the results section indicate that the results obtained by the algorithm are satisfactory, but some minor inconsistencies were detected. However, in order to be able to draw more solid conclusions about the quality of the algorithm, it would be needed to do more extensive experiments.

Appendix

Here, we will prove that the cut function on a weighted graph $G = (V, E)$ with positive weights is indeed submodular.

Let F be the cut function, let A and B be two sets such that $A \subseteq B \subseteq V$ and let $s \in V$ be an element such that $s \notin B$. Now it holds that:

$$F(A \cup \{s\}) = F(A) - \sum_{v \in A} w_{v,s} + \sum_{v \in V \setminus A} w_{s,v} \quad (4)$$

Similarly for B we have:

$$F(B \cup \{s\}) = F(B) - \sum_{v \in B} w_{v,s} + \sum_{v \in V \setminus B} w_{s,v} \quad (5)$$

The desired inequality, needed for submodular functions, is:

$$F(A \cup \{s\}) - F(A) \geq F(B \cup \{s\}) - F(B) \quad (6)$$

This inequality holds if and only if (fill in equation 4 and equation 5):

$$-\sum_{v \in A} w_{v,s} + \sum_{v \in V \setminus A} w_{s,v} \geq -\sum_{v \in B} w_{v,s} + \sum_{v \in V \setminus B} w_{s,v} \quad (7)$$

Because $A \subseteq B$, it holds that

$$-\sum_{v \in A} w_{v,s} \geq -\sum_{v \in B} w_{v,s} \quad (8)$$

and

$$\sum_{v \in V \setminus A} w_{s,v} \geq \sum_{v \in V \setminus B} w_{s,v} \quad (9)$$

such that the desired inequality indeed holds.

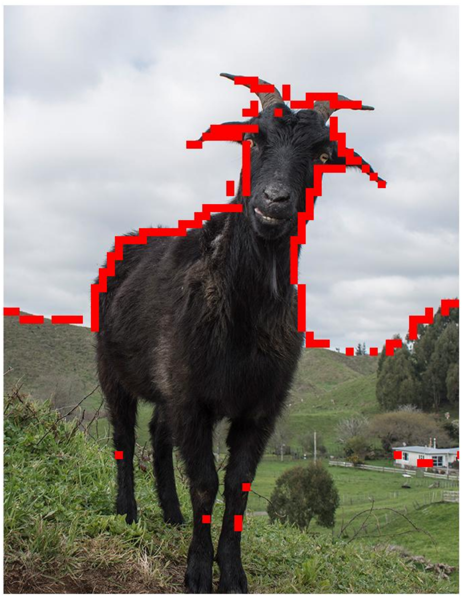

Figure 8: The result of running our algorithm on the example in figure 6 with $\alpha = 0.5$.

References

[1] Francis R. Bach. Learning with submodular functions: A convex optimization perspective. *CoRR*, abs/1111.6453, 2011.

[2] Jorik Jooken and Michiel Baptist. Finding image structure using submodularity. https://github.com/MichielBaptist/Structure-Finding-Submodularity.

[3] Andreas Krause. Sfo: A toolbox for submodular function optimization. *J. Mach. Learn. Res.*, 11, 03 2010.

[4] Andreas Krause and Carlos Guestrin. Beyond convexity - submodularity in machine learning. tutorial, 2008.

[5] Sebastian Ruder. An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.04747, 2016.