Paint Regression Project Mathematical Description

Alon Jacobson

January 22, 2022

1 Introduction

The goal of my paint regression project is to try to get the computer to "paint" an image, by using a highly simplified model of how an artist paints a picture: this is how one would opt to paint if they wanted to get their painting as close as possible to a reference image with the least amount of effort. The goal of the program is to create a painting that is as close as possible to the source image after a given fixed number of brush strokes. A "brush stroke" is a region of pixels in the image, together with a color and an opacity, and applying a brush stroke is coloring in the painting with the color and opacity in the region.

2 Setup

We have a source image I, represented as an array of pixels with m rows, n columns, and c channels (c will usually be 3 for red, green, and blue). Color intensities are from 0 to 1. So, $I \in \mathcal{I}$, where $\mathcal{I} = [0,1]^{m \times n \times c}$ is the set of all possible images. The program is trying to recreate I by placing brush strokes on a painting one-by-one. The painting after t brush strokes is denoted $P_t \in \mathcal{I}$. The painting can start out as anything, but in my program I start the painting as being white to represent a blank canvas, so we can set P_0 to be the filled with 1s.

Represent a brushstroke as a triple $B = (R, \vec{C}, O)$.

$R \in \mathcal{R}$ is the region in the image that the brush paints on, where $\mathcal{R}$ is the set of all possible regions. If R is represented as a set of pixel positions, then set of all regions $\mathcal{R}$ is the set of all sets of pixel positions, i.e., the set of all subsets of the set of all pixel positions. Denote $\mathcal{P} = \{1, \ldots, m\} \times \{1, \ldots, n\}$ as the set of all pixel positions of the image. The set of all regions, then, is the power set of $\mathcal{P}$, denoted $2^{\mathcal{P}}$, so in this case $\mathcal{R} = 2^{\mathcal{P}}$.

$\vec{C} \in \mathcal{C}$ is the color of the brush, where $\mathcal{C}$ is the set of all colors and $\mathcal{C} = [0,1]^c$.

$O \in \mathcal{O}$ is the opacity of the brush, where $\mathcal{O}$ is the set of all opacities, and $\mathcal{O} = [0,1]$.

The "loss" of the painting after t iterations, $\mathcal{L}(P_t)$, is the total squared error between the

painting and the image. We have a function $\mathcal{L} : \mathcal{I} \to \mathbb{R}$ which gives the loss of a painting:

$$\begin{aligned}\mathcal{L}(P) &:= \sum_{i=1}^{m}\sum_{j=1}^{n}\sum_{k=1}^{c}(P(i,j,k) - I(i,j,k))^2 \\ &= \sum_{i=1}^{m}\sum_{j=1}^{n}\underbrace{\|P(i,j) - I(i,j)\|^2}_{\ell_P(i,j)}\end{aligned}$$

With each iteration, a brush stroke is chosen to try to minimize the loss at the next iteration.

Let $B_t = (R_t, \vec{C}_t, O_t)$ be the brushstroke used at time t. The painting at iteration t is given by:

$$P_t = T_{B_t}(P_{t-1})$$

where $T_{R,\vec{C},O}$ is a parameterized function that takes in an image returns that image with a brush stroke applied to it:

$$[T_{R,\vec{C},O}(P)](i,j) = \begin{cases} b_{\vec{C},O}(P(i,j)) & (i,j) \in R \\ P(i,j) & \text{otherwise} \end{cases}$$

where

$$b_{\vec{C},O}(\vec{x}) = O\vec{C} + (1-O)\vec{x}.$$

Now I will write change in the loss from applying a brush $B = (R, \vec{C}, O)$. I will call this change of loss $\Delta\mathcal{L}_P(B) := \mathcal{L}(T_B(P)) - \mathcal{L}(P)$. Write $P' = T_B(P)$ as shorthand. First, note that

$$\sum_{(i,j)\notin R}\ell_{P'}(i,j) = \sum_{(i,j)\notin R}\|P'(i,j) - I(i,j)\|^2 = \sum_{(i,j)\notin R}\|P(i,j) - I(i,j)\|^2 = \sum_{(i,j)\notin R}\ell_P(i,j).$$

$$\begin{aligned}\Delta\mathcal{L}_P(B) = \mathcal{L}(T_B(P)) - \mathcal{L}(P) &= \sum_{i,j}\ell_{P'}(i,j) - \sum_{i,j}\ell_P(i,j) \\ &= \sum_{(i,j)\in R}\ell_{P'}(i,j) + \sum_{(i,j)\notin R}\ell_{P'}(i,j) - \sum_{(i,j)\in R}\ell_P(i,j) - \sum_{(i,j)\notin R}\ell_P(i,j) \\ &= \sum_{(i,j)\in R}\ell_{P'}(i,j) + \sum_{(i,j)\notin R}\ell_P(i,j) - \sum_{(i,j)\in R}\ell_P(i,j) - \sum_{(i,j)\notin R}\ell_P(i,j) \\ &= \sum_{(i,j)\in R}\ell_{P'}(i,j) - \sum_{(i,j)\in R}\ell_P(i,j) \\ &= \sum_{(i,j)\in R}\|P'(i,j) - I(i,j)\|^2 - \sum_{(i,j)\in R}\ell_P(i,j) \\ &= \sum_{(i,j)\in R}\|b_{\vec{C},O}(P(i,j)) - I(i,j)\|^2 - \sum_{(i,j)\in R}\|P(i,j) - I(i,j)\|^2.\end{aligned}$$

This formula avoids the need to sum over the whole image, instead just summing over the region painted on. Note that the function $\Delta\mathcal{L}_P$ always outputs a non-positive value.

Suppose we have a painting P, and we have a set of brushes $S \subseteq \mathcal{R} \times \mathcal{C} \times \mathcal{O}$ to choose from. We want to choose the brush $B \in S$ that minimizes $\mathcal{L}(T_B(P))$. This is equivalent to minimizing $\Delta\mathcal{L}_P(B)$, since $\Delta\mathcal{L}_P(B)$ and $\mathcal{L}(T_B(P))$ differ by $\mathcal{L}(P)$, which is constant for all brushes. Thus when finding brushes we will minimize $\Delta\mathcal{L}_P$.

3 Choosing color and opacity given a brush region

Suppose that we have the current painting P to paint on, and we already have a chosen region to paint on, R. Then we can choose the color $\vec{C}$ and opacity O that minimize $\Delta\mathcal{L}_P(R, \vec{C}, O)$. As shown above,

$$\Delta\mathcal{L}_P(B) = \sum_{(i,j)\in R} \|b_{\vec{C},O}(P(i,j)) - I(i,j)\|^2 - \sum_{(i,j)\in R} \|P(i,j) - I(i,j)\|^2$$

and the second term is constant with respect to $\vec{C}$ and O, so we want to minimize the following quantity:

$$\sum_{(i,j)\in R} \|b_{\vec{C},O}(P(i,j)) - I(i,j)\|^2 \tag{1}$$

And if we write $\vec{\alpha} = O\vec{C}$ and $\beta = 1 - O$, then

$$b_{\vec{C},O}(\vec{x}) = O\vec{C} + (1-O)\vec{x} = \vec{\alpha} + \beta\vec{x}$$

and it doesn't matter if we optimize $\vec{\alpha}$ and β instead of $\vec{C}$ and O because you can go back and forth between the two forms. If we write $R = \{(i_1, j_1), \ldots, (i_N, j_N)\}$, and then write

$$X := \begin{bmatrix} P(i_1, j_1) \\ \vdots \\ P(i_N, j_N) \end{bmatrix}, \quad Y := \begin{bmatrix} I(i_1, j_1) \\ \vdots \\ I(i_N, j_N) \end{bmatrix},$$

then we can rewrite (1):

$$\sum_{(i,j)\in R} \|b_{\vec{C},O}(P(i,j)) - I(i,j)\|^2 = \sum_{i=1}^{N} \|\vec{\alpha} + \beta X_i - Y_i\|^2 = \sum_{i=1}^{N}\sum_{j=1}^{c} [\vec{\alpha}_j + \beta X_{i,j} - Y_{i,j}]^2.$$

We want to choose $\vec{\alpha}$ and β that minimize this quantity. By taking partial derivatives, one can obtain the following solution $(\vec{\alpha}^*, \beta^*)$.

Write the mean of the k component of X and Y as

$$\overline{X_{:,k}} := \frac{1}{N}\sum_{i=1}^{N} X_{i,k}, \quad \overline{Y_{:,k}} := \frac{1}{N}\sum_{i=1}^{N} Y_{i,k},$$

and S_{xx} and S_{xy} are

$$S_{xx} := \sum_{i=1}^{N}\sum_{j=1}^{c} (X_{i,j} - \overline{X_{:,j}})^2, \quad S_{xy} := \sum_{i=1}^{N}\sum_{j=1}^{c} (X_{i,j} - \overline{X_{:,j}})(Y_{i,j} - \overline{Y_{:,j}})$$

then the solution is:

$$\alpha_k^* = \overline{Y_{:,k}} - \beta^* \overline{X_{:,k}},$$

and

$$\beta^* = \frac{S_{xy}}{S_{xx}} \quad \text{if } S_{xx} \neq 0.$$

If $S_{xx} = 0$, then it must be the case that $S_{xy} = 0$ in which case any value of β^* works (so if $S_{xx} = 0$ but $S_{xy} \neq 0$, then there was an arithmetic error).

Once we obtain $(\vec{\alpha}^*, \beta^*)$ which are optimal, we can set $O^* = 1 - \beta^*$, $\vec{C}^* = \frac{\vec{\alpha}^*}{O^*}$. If $O^* \notin \mathcal{O} = [0, 1]$ or $\vec{C}^* \notin \mathcal{C} = [0, 1]^c$, then we could try to do a constrained optimization, but I'm lazy so in this case let's just reject the region R and pick another region.

4 Choosing a brush region

Now that we are able to pick the optimal brush color and opacity given a brush region, we are concerned with picking the best brush region. There are way too many regions to check by brute force, and you can't differentiate with respect to the region to try to do gradient based optimization, so what we will use random-restart hill climbing.

To generate random neighbors, "parameterized brush regions" will be used. We have a number of parameterized brush region classes, labelled 1 through K. Each parameterized brush region class $k \in \{1, \ldots, K\}$ has associated with it a set of possible parameters Θ_k (usually $\Theta_k \subseteq \mathbb{R}^p$ for some $p \in \mathbb{N}$), and a brush in that class is identified by its parameters. The class also has a random variable $G_k : \Omega_k \to \Theta_k$ which generates random parameters and is to be used by random search and random-restart hill climbing; a function $r_k : \Theta_k \to \mathcal{R}$ which returns the region of a brush given its parameters; and a function $n_k : \Theta_k \to \Theta_k$ which generates a "neighbor" of a parameterized region and is to be used by hill climbing.

Let's make a function that given a region and a painting, gives the optimal color and opacity: $p : \mathcal{R} \times \mathcal{I} \to \mathcal{C} \times \mathcal{O}$ and $p(R, P) = (\vec{C}^*, O^*)$.

5 Algorithm

The basic algorithm is as follows (note that in step 2(c)i there is an anonymous function using the lambda calculus notation):

1. Start with P_0 having all ones (white).
2. For iteration $t = 0$ to max iter:
   (a) Set $L_{\text{best}} = 0, (k_{\text{best}}, \theta_{\text{best}}) = None$
   (b) For each brush class $k = 1$ through K:
   (c) i. $(\theta, L) :=$ HillClimbing$(\lambda\theta.\Delta\mathcal{L}_{P_t}(r_k(\theta), p(r_k(\theta), P_t)), G_k, n_k)$
   ii. If $L < L_{\text{best}}$, then
   - $(k_{\text{best}}, \theta_{\text{best}}) := (k, \theta)$
   - $L_{\text{best}} := L$

(d) $P_{t+1} := T_{r_{k_{\text{best}}}(\theta_{\text{best}}),p(r_{k_{\text{best}}}(\theta_{\text{best}}),P_t)}(P_t)$