THE POON-DOMINGOS PARAMETER LEARNING ALGORITHM FOR IMAGE COMPLETION AND CLASSIFICATION ON SUM-PRODUCT NETWORKS

Renato Lui Geh
Computer Science
Institute of Mathematics and Statistics
University of São Paulo
renatolg@ime.usp.br

Abstract. In this document we describe the Poon-Domingos [PD11] parameter learning algorithm for image classification and completion.

1. Structure

The Poon-Domingos algorithm uses a fixed structure and then learns the weights through generative learning. We first give an overview on how to build the structure given an image and then provide a pseudo-code algorithm for building such structure. In this document we assume instances as images. However, the Poon-Domingos structure allows for any object with local dependencies.

1.1. Overview

The Poon-Domingos structure models a probability distribution over a set of variables with local dependencies. On the plain, one could argue it models rectangular neighborhoods for each point in the space. In the article [PD11], Poon and Domingos use images as a dataset, with dependencies being rectangular pixel neighborhoods. Images are an example of local dependencies, since a pixel has possible dependencies with their neighbors.

Dennis and Ventura explain an intuition of how the Poon-Domingos structure algorithm works [DV12]. We expand on this intuition, giving insights on how such an algorithm is built and showing a pseudo-code visualization of it. Once we have shown how to build the SPN structure, we describe generative learning through gradient descent, and later expectation-maximization.

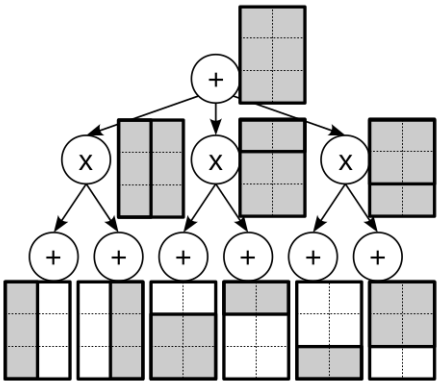

FIGURE 1. The Poon architecture with $r = 1$ resolution and $k = 1$ sum nodes per region on a 2×3 image. At each r resolution axis-aligned rectangular decomposition, we create k sum nodes.[DV12]

1.2. Definitions and properties

Definition 1.1 (Region)**.** *A Region $\mathcal{R}$ is a rectangular part of an image. Let $p_0 = (x_0, y_0)$ and $p_1 = (x_1, y_1)$ be the top-left and bottom-right pixels of $\mathcal{R}$ relative to the image. These are called the coordinates of $\mathcal{R}$.*

Definition 1.2 (Region Node)**.** *A Region Node R has a one-to-one and onto mapping with a Region $\mathcal{R}$. R has k internal nodes associated with it. If $\mathcal{R}$ is over an $r \times r$ set of pixels (i.e. the atomic unit), then R has k leaf nodes (e.g. k-mixture of gaussians). Else, R has k sum nodes.*

Definition 1.3 (Decomposition)**.** *Let $\mathcal{R}$ be a Region. A Decomposition $\mathcal{D}$ is an axis-aligned partitioning of $\mathcal{R}$ into two Regions $\mathcal{R}_1$ and $\mathcal{R}_2$.*

The decomposition $\mathcal{D}$ of a Region $\mathcal{R}$ involves a few steps. Let $\mathcal{R}_1$ and $\mathcal{R}_2$ be the resulting subregions product of the decomposition. The resulting subgraph of the SPN S of this decomposition is a DAG G. If $\mathcal{R}$ is the entire image, then the root of G is a single sum node and $G = S$. Otherwise, then the root of G is a region node and thus the root of G is a set of k sum nodes. Let R be the root node of G. Region nodes R_1 and R_2 will both have k sum nodes (or univariate distributions for leaves). We shall denote as R_i^j the j-th sum node of region node i. For each pairing of sum nodes (R_1^i, R_2^j), we create a product node π and add R_1^i and R_2^j as children of π. The set Π of these product nodes are the decomposition nodes of $\mathcal{R}$ into $\mathcal{R}_1$ and $\mathcal{R}_2$. Once we have created all product nodes in this set, we add all of them as children of R. If R is a region node, then adding Π as children of R means, for every sum node σ in R, set product node $\pi \in \Pi$ as a child of σ. Each product set Π will have $\binom{m}{2}$ product nodes, since we take every distinct pairing of sum nodes in R_1 and R_2.

Since a region node R is unique, we may have different decompositions in which the same region appears more than once. For this reason we should create a single

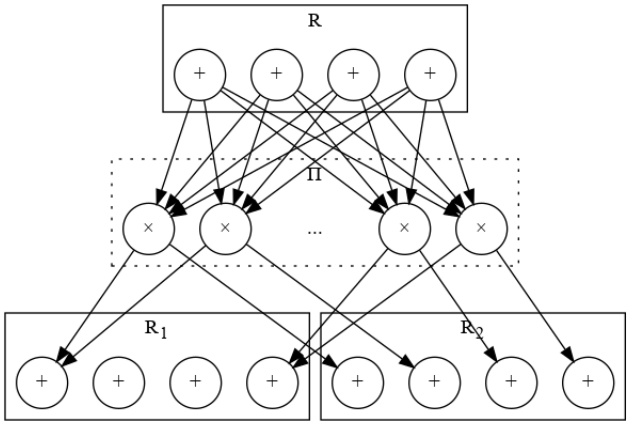

FIGURE 2. A decomposition of a Region $\mathcal{R}$ into two subregions $\mathcal{R}_1$ and $\mathcal{R}_2$. The set Π of product nodes are the decomposition nodes that connect the unsplit image to the partitions. Each element $\pi \in \Pi$ connects a pairing of a sum node of $\mathcal{R}_1$ and of $\mathcal{R}_2$.

Region Node for each possible region. We need a map function that takes the top-left and bottom-right pixel positions of a region and maps it to a number for storage. Since every region is unique, we need a one-to-one and onto function.

Definition 1.4 (Region map function)**.** *A region map function is a function that maps a region into an integer. We define it as*

$$f : \mathbb{Z}_m \times \mathbb{Z}_n \times \mathbb{Z}_m \times \mathbb{Z}_n \to \mathbb{Z}_{m^2n^2}$$
$$f(x_1, y_1, x_2, y_2) = ((y_1 m + x_1)m + x_2)n + y_2$$

where $x_1, x_2 \in \mathbb{Z}_m$ *and* $y_1, y_2 \in \mathbb{Z}_n$.

Proposition 1.1. *The region map function is one-to-one and onto.*

Proof. We first prove f is one-to-one. If f is injective, then $f(x_1, y_1, x_2, y_2) = f(x_1', y_1', x_2', y_2') \Rightarrow (x_1, y_1, x_2, y_2) = (x_1', y_1', x_2', y_2')$. Suppose $f(x_1, y_1, x_2, y_2) = f(x_1', y_1', x_2', y_2')$ for some $x_i \in \mathbb{Z}_m$ and $y_i \in \mathbb{Z}_n$. Then we have:

$$((y_1 m + x_1)m + x_2)n + y_2 = ((y_1' m + x_1')m + x_2')n + y_2'$$
$$(m^2 y_1 + m x_1 + x_2)n + y_2 = (m^2 y_1' + m x_1' + x_2')n + y_2'$$
$$m^2 n y_1 + mn x_1 + n x_2 + y_2 = m^2 n y_1' + mn x_1' + n x_2' + y_2'$$
$$m^2 n(y_1 - y_1') + mn(x_1 - x_1') + n(x_2 - x_2') + (y_2 - y_2') = 0$$

But $m, n > 0$. Therefore, it is easy to see that $x_i - x_i' = 0$ and $y_i - y_i' = 0$ is necessary for the equation to hold. Proof of surjection is simple. Since we know f is one-to-one and that $\mathbb{Z}_m \times \mathbb{Z}_n \times \mathbb{Z}_m \times \mathbb{Z}_n$ has the same number of elements as $\mathbb{Z}_{m^2n^2}$, than it follows that f must be onto. □

Bijection of the region map function is necessary since we need the inverse function f^{-1} to be symmetrical to f. That is, we must be able to encode a region into

a number and later be able to find what region a number represents. We define the inverse function of f below.

Definition 1.5 (Inverse region map function)**.** *The inverse of the region map function is given by the decomposition of an integer $r \in \mathbb{Z}_{m^2n^2}$ into a tuple $(x_1, y_1, x_2, y_2) \in \mathbb{Z}_m \times \mathbb{Z}_n \times \mathbb{Z}_m\mathbb{Z}_n$. Let $g = f^{-1}$. We define g as an algorithm as follows*

Algorithm 1 Function `Encode` $\coloneqq f^{-1} = g$

Input $r \in \mathbb{Z}_{m^2n^2}$
Output $(x_1, y_1, x_2, y_2) \in \mathbb{Z}_m \times \mathbb{Z}_n \times \mathbb{Z}_m \times \mathbb{Z}_n$
1: $y_2 \leftarrow i \mod n$
2: Let $c \in \mathbb{Z}_{m^2n^2}$
3: $c \leftarrow \frac{(r-y_2)}{n}$
4: $x_2 \leftarrow c \mod m$
5: $c \leftarrow \frac{c-x_2}{m}$
6: $x_1 \leftarrow c \mod m$
7: $y_1 \leftarrow \frac{c-x_1}{w}$
8: **return** (x_1, y_1, x_2, y_2)

1.3. Structural algorithm

We now show how to construct the Poon-Domingos architecture given an image I (which is equivalent to a Region consisting of the entire image), resolution r and m sum nodes per region.

The algorithm, as summarized in the last subsection, can be constructed recursively. However we avoid this technique in favor of an iterative version, which uses less memory. We must first do a preprocessing of Regions. We iterate over all possible subrectangles in I, create a Region for each of them and assign an identification number to it. This number is the value of the region map function given the region's position. We name the region map function as `Encode` that takes a region position and returns a non-negative integer. Similarly, we name the inverse function of `Encode` as `Decode` that takes a non-negative integer and returns a region position.

We denote by $[\mathcal{R}]$, where $\mathcal{R}$ is a region, a function that returns a pair of positive integers that represent the width and height of $\mathcal{R}$. Once we have constructed all possible regions, we iterate over all possible decompositions according to the following steps:

1. Select each possible region $\mathcal{R}$ in image I
2. If $\mathcal{R} = I$, then R is a sum node and is root
3. Else if R contains only gaussians, skip this region
4. Else, R exists and is a set of sum nodes
5. Partition $\mathcal{R}$ into a pairing of subregions $\mathcal{R}_1$ and $\mathcal{R}_2$
6. For each subregion $\mathcal{R}_i$
   6.1 Let $(w, h) \leftarrow \mathcal{R}_i$

6.2 If $w > r$ and $h > r$ then region node R_i must be a set of m gaussians
6.3 Else, region node R_i must be a set of m sum nodes
7. Create a set Π of product nodes
8. For each pair (R_1^i, R_2^j), where R_p^q is the inner node q of subregion node R_p
8.1 Create a product node π and add it to set Π
8.2 Add R_1^i and R_2^j as children of π
8.3 Add π as child of all inner sum nodes of R
9. Go to step 1.

Algorithm 2 `CreateRegions`

Input A pair (w, h) representing the image I dimensions
Input Dataset $\mathcal{D}$
Input Integer m as number of components in each Region
Input Integer r as resolution
Output A list $\mathcal{L}$ of Nodes indexed by their region map function value

Let $n = w \cdot h$
for $i \leftarrow 0..(n-1)$ **do**
    $x_1 \leftarrow i \mod w$
    $y_1 \leftarrow i/w$
    for $x_2 \leftarrow (w-1)..x_1$ decreasingly **do**
        for $y_2 \leftarrow (h-1)..y_1$ decreasingly **do**
            if $x_1 = 0$ and $y_1 = 0$ and $x_2 = w-1$ and $y_2 = h-1$ **then**
                ▷ The entire image is the root sum node
                $j \leftarrow$`Encode`(x_1, y_1, x_2, y_2)
                $\mathcal{L}[j] \leftarrow$`SumNode`()
                ▷ `SumNode` returns a sum node
            end if
            Let R be a new Region node
            Let $d_x = x_2 - x_1$ and $d_y = y_2 - y_1$
            if $d_x < r$ or $d_y < r$ **then**
                $x \leftarrow \max(x_1 + r, x_2)$
                $y \leftarrow \max(y_1 + r, y_2)$
                $r \leftarrow$`Encode`(x_1, y_1, x, y)
                $R \leftarrow \mathcal{L}[r]$
            else if $d_x = r$ and $d_y = r$ **then**
                $R \leftarrow$`GaussianMixture`$(x_1, y_1, x_2, y_2, D, m)$
                ▷ `GaussianMixture` returns a node containing m gaussians
            else
                $R \leftarrow$`RegionNode`(m)
                ▷ `RegionNode` returns a node containing m sum nodes
            end if
            $j \leftarrow$`Encode`(x_1, y_1, x_2, y_2)
            $\mathcal{L}[j] \leftarrow R$
        end for
    end for
end for
return $\mathcal{L}$

The structural algorithm is almost complete. The main object of interest now is on finding all possible regions in an image. Consider the matrix $M_{n\times m}$ as the image representation of I indexed by 0. Then we can find all possible regions by iteration over every top-left position (x, y) and for each of these positions iterating over all bottom-right pixels $(x + p, y + q), 0 \leq p < n, 0 \leq q < m$.

We assume a dataset $\mathcal{D}$ where each instance $\mathcal{D}[X]$ gives an ordered sequence of integers corresponding to the image pixels in expanded form. Let R be a region, (x_1, y_1) be its top-left position and (x_2, y_2) its bottom-right position. We call (x_1, y_1, x_2, y_2) the coordinates of R.

Proposition 1.2. *Let R_1 be the region in Line 13. Line 19 in function* `CreateRegions` *always yields a region R_2 that has already been previously created.*

Proof. Let (x_1, y_1) be R_1's top-left position. We fix $x_2 \geq x_1 + r$. The third `for` loop in `CreateRegions` yields a decreasing sequence from $h - 1$ to y_1. So for each y_2 in this ordered sequence, the previous instance will have already been computed. It then follows that for every $y_2 = y_1 + r - k$, $0 \leq k \leq r$, it is true that $y_1 + r$ has already been computed. We do the same for x_2 by fixing $y_2 \geq y_1 + r$. When both $x_2 < x_1 + r$ and $y_2 < y_1 + r$, we fall into the previous subcases. □

Algorithm 3 `Structure`

Input A pair (w, h) representing the image I dimensions
Input Dataset $\mathcal{D}$
Input Integer m as number of components in each Region
Input Integer r as resolution
Output An SPN S containing the learned structure from image I

1: $\mathcal{L} \leftarrow$`CreateRegions`(w, h, D, m, r)
2: $s \leftarrow$`Encode`$(0, 0, w - 1, h - 1)$
3: $S \leftarrow \mathcal{L}[s]$
4: $\triangleright$ S is the root node
5: **for** each key and value pair (k, R) in $\mathcal{L}$ **do**
6: **if** $k = s$ **then**
7: **skip**
8: **end if**
9: $(x_1, y_1, x_2, y_2) \leftarrow$ `Decode`(k)
10: `LeftQuadrant`$(x_1, y_1, x_2, y_2, \mathcal{L})$
11: `BottomQuadrant`$(x_1, y_1, x_2, y_2, \mathcal{L})$
12: **end for**
13: **return** S

Next, we clarify functions `LeftQuadrant` and `BottomQuadrant`. Recall the algorithm outline we listed earlier. For each decomposition of a region into two subregions, we must create a set of product nodes whose internal products nodes are connected the two distinct subregion internal nodes. Let R be a region. Our goal is to find the parent regions of R. In other words, we must find all possible decompositions in which R is one of the two disjoint subregions. Since we only take

axis-aligned decompositions into account, we can represent all possible decompositions of R as tightly-bound subrectangles.

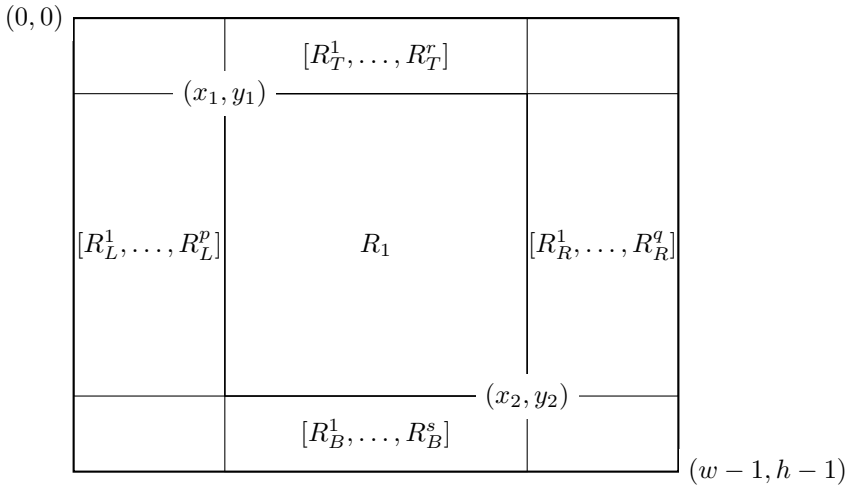

FIGURE 3. Given a decomposition of a region into two subregions, where one of them is R_1, we can find any possible complementary regions of R_1 by searching each quadrant. The image above shows all possible decompositions containing R_1, where the set $[R_Q^1, \ldots, R_Q^m]$, where Q is a quadrant, contains all complementary regions of R_1 in the direction of Q. A decompositon is two regions R_1 and R_Q^i.

To find all the decompositions that contains a region R_1, we take each possible quadrant (we name them $\{L, R, T, B\}$ for Left, Right, Top and Bottom) and then take all possible R_Q^i region, where Q is the quadrant and i the i-th region in Q. It suffices to find decompositions from L and B, as we will eventually do the same for the complementary region of R_1, which will account for the R and T quadrants.

Figure 4 shows a decomposition that contains subregions R_1 and R_2. Let (x_1, y_1, x_2, y_2) be the coordinates to R_1. We have x_1 subregions R_2 that are possible decompositions with R_1. Similarly for Figure 5, we have y_1 possible subregions R_2. Consider the left quadrant. If R_1 switches place with R_2, it becomes clear that we now have the right quadrant. Similarly for the bottom quadrant, by switching R_1 and R_2, we have the top quadrant. This means our previous assumption that it is enough to take the left and bottom quadrant is correct.

At each pairing of subregions R_1 and R_2 in a quadrant Q, we must create a set of product nodes Π.

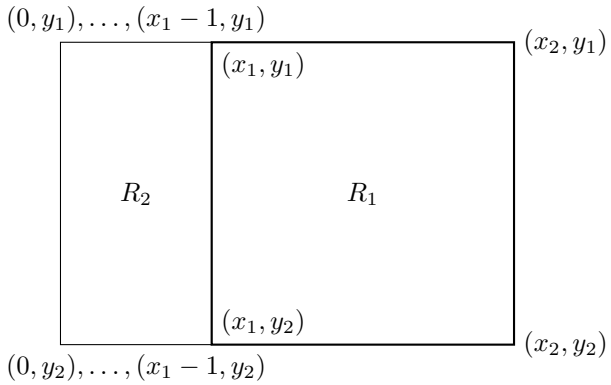

FIGURE 4. The left quadrant. Region R_2 is the complementary region of R_1 and can take any coordinates (x', y_1, x_1, y_2), where $0 \leq x' \leq x_1 - 1$.

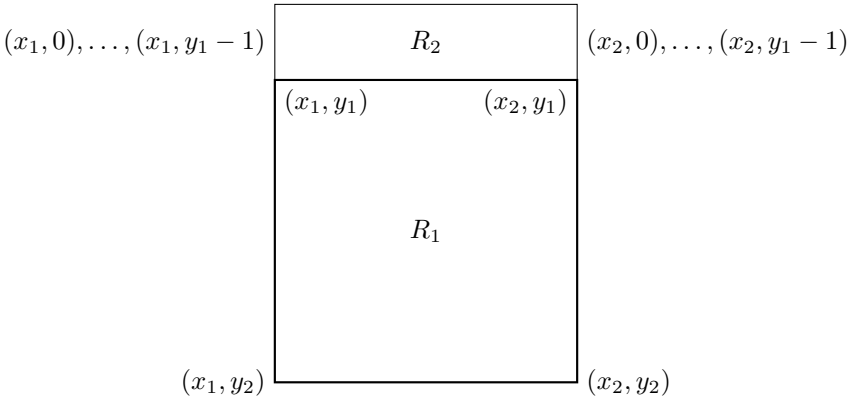

FIGURE 5. The left bottom. Region R_2 is the complementary region of R_1 and can take any coordinates (x_1, y', x_2, y_1), where $0 \leq y' \leq y_1 - 1$.

REFERENCES

[DV12] Aaron Dennis and Dan Ventura. "Learning the Architecture of Sum-Product Networks Using Clustering on Variables". In: *Advances in Neural Information Processing Systems* 25 (2012).

[PD11] Hoifung Poon and Pedro Domingos. "Sum-Product Networks: A New Deep Architecture". In: *Uncertainty in Artificial Intelligence* 27 (2011).