Intro to Image Understanding (CSC420) Assignment 5

Posted: Nov 21, 2017
Submission Deadline: Dec 7 (Thu), 11:59, 2017
Point of contact TA: iab@cs.toronto.edu

Instructions for submission: Please write a document and submit a **PDF** with your solutions (include pictures where needed). Include your code inside the document, and submit through **MarkUs**. You should also upload the output of your tracking system as a zip alongside your PDF, as detailed in Exercise 1.2.

For full marks you must show your work, not just your final answer.

Max points: 12, max extra credit points: 3

1 Tracking (7 points)

You are given a short video clip broken down into consecutive frames. For each frame you are also provided with a set of detection boxes for the person class obtained via the Deformable Part-based Model (DPM). The data is structured in the following way:

- The frames are in `data/frames`.
- The detections are in `data/detections`. Each frame has its own mat file which contains a variable `dets`. This variable has the same structure as `ds` in the `demo_car` function in Assignment 4. Thus, each row of `dets` is `[left, top,right, bottom, id, score]`, specifying the coordinates of the detection box, and its confidence (score).

Your task is to complete a function called `code/track_objects.m` (MATLAB users) or `code/track_objects.py` (Python users). This function loads detections of two consecutive frames, and computes similarity between each detection in frame i and each detection in frame $i+1$. Your goal is to find an assignment between detections in consecutive frames of the videos, that are believed to correspond to the same, possibly moving, object. The assignment between detections in multiple frames is called a *track*. There are several different options of how to do that, but for now we will focus on a simple yet effective greedy method.

1.1. (3 points) Greedy tracking Initialize tracks as an empty list. Visit each frame. Assign two detections (one in the current and one in the next frame) with the highest similarity to the same track and add the track to the track list. Pick the next two most similar detections (one in the current and one in the next frame, both detections should be disjunct from the already selected pair), add them to the next track, etc. When no more disjunct pairs of detections exist, move on to the next frame. Again pick two detections (one in the current and one in the next frame) with the highest similarity. If one of the tracks in the existing track list contains any of the picked detections, then just assign the new detection to this track. If not, add the new pair as a new track. Pick the next pair of most similar detections, etc. Once you visit all the frames, remove tracks that contain only two detections (this is probably noise).

In your solution document, include a short explanation of your method. Include also code (the completed track objects function) plus any other function you may have written for this purpose.

1.2. (2 points) Solution Visualization Plot each frame with all detections (boxes) that have been tracked for more than 5 frames. Each different track should have a rectangle (box) of different color, however all detections corresponding to the same track should have the same color. Store a visualization of each frame in a directory called tracks. You do not need to insert the frames into your solution

document. Simply include the tracks directory in a zip file along with your document and upload to MarkUs.

1.3. (1 point) Do not worry if you don't track all the players. Some of them may not be detected in all the frames. Any idea how to deal with missing detections? No need to write code, just write down your idea.

1.4. (1 point) Among all your tracks how would you find the soccer player that was running the fastest? Be careful with your answer: a player very far away seems to move less in an image, but the player may in fact have been running like Flash. You do not need to provide any code, just a written answer will do.

2 Computation Graphs and Deep Learning (5 points)

Note: to solve this problem you will need to review slides from Lecture 15—"Basics of Training Neural Networks."

In the lecture we covered a building block of neural nets, the logistic regression. Given M training examples, the goal of logistic regression is to maximize the following probability w.r.t. $\mathbf{w}$ and b:

$$P(\mathbf{y}|\mathbf{w}, b, \mathbf{X}) = \prod_{i=1}^{M} \left(h(\mathbf{w}^\top \mathbf{x}_i + b)\right)^{y_i} \left(1 - h(\mathbf{w}^\top \mathbf{x}_i + b)\right)^{1-y_i}, \tag{1}$$

where $\mathbf{x}_i \in \mathbb{R}^n$ is an n-dimensional feature vector for the i^{th} training example, and $y_i \in \{0, 1\}$ is the *binary* class label of that i^{th} example. Additionally, h represents the *logistic* (also called the *sigmoid*) function,

$$h(\mathbf{w}^\top \mathbf{x}_i + b) = \frac{1}{1 + e^{-(\mathbf{w}^\top \mathbf{x} + b)}}. \tag{2}$$

Often, when exponentials are involved in the expression of a probability, it is customary to optimize the log of the probability. Applying the log function also has the added benefit of transforming the large product from Equation 1 into a sum. This is a desirable property, as it is typically easier to work with sums than it is with products. Moreover, large products are also difficult to deal with in software, as multiplying many probabilities (i.e., values smaller than or equal to one) can quickly underflow floating point number representations. Adding up logs does not exhibit this problem. After a computation is complete, its output can always be exponentiated in order to return to the domain of probabilities.

In addition to the aforementioned application of the logarithm, it is also more common, by convention, to minimize a loss, rather than maximize an objective. Therefore, the usual loss for logistic regression is formulated as

$$\mathcal{L}(\mathbf{w}, b) = -\frac{1}{M} \sum_{i=1}^{M} \left[y_i \log\left(h(\mathbf{w}^\top \mathbf{x}_i + b)\right) + (1 - y_i) \log\left(1 - h(\mathbf{w}^\top \mathbf{x}_i + b)\right) \right], \tag{3}$$

and the optimization objective becomes

$$(\mathbf{w}^*, b^*) = \arg\min_{\mathbf{w}, b} \mathcal{L}(\mathbf{w}, b). \tag{4}$$

Let us now consider the simple case with **only two features**, i.e., where you only have three parameters to optimize for: $w_1, w_2, b \in \mathbb{R}$.

2.1. (1.5 points) Derive the analytical expressions for the following derivatives:

$$\frac{\partial \mathcal{L}}{\partial w_1}, \quad \frac{\partial \mathcal{L}}{\partial w_2}, \quad \frac{\partial \mathcal{L}}{\partial b} \tag{5}$$

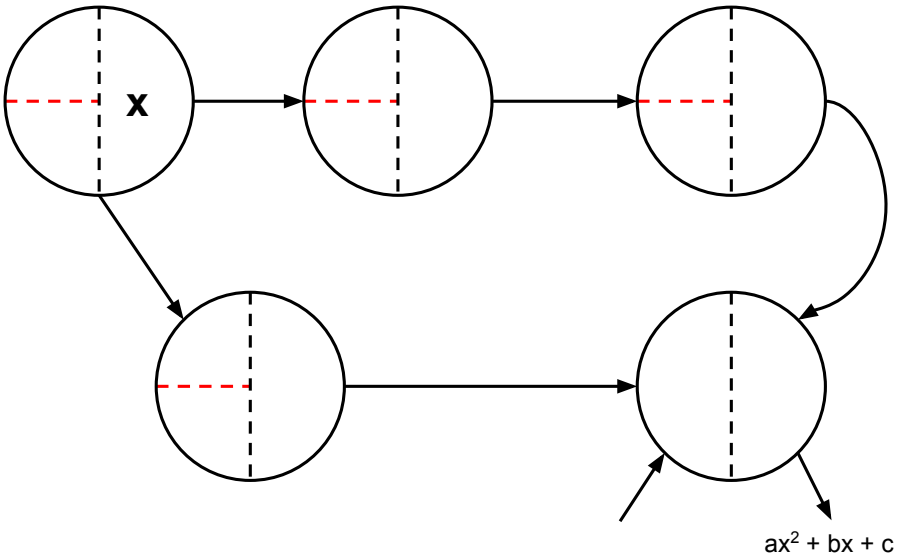

Figure 1: A blank computation graph which should be filled out using the quadratic function example in the lecture.

2.2. (0.5 points) As a warm-up exercise, recall the example computation graph for the quadratic function ax^2+bx+c covered in class. Fill out the blank graph provided in Figure 1 with the intermediary steps which were left out, such that the final derivative you get when you "return" to the x node is the correct one, i.e., $2ax+b$. Don't just copy the example from the lecture! Try to work through it yourself, such that you get a hang of how to apply the chain rule using computation graphs. It will come in handy for the next exercises.

2.3. (1 point) Using the ideas introduced in Lecture 15 (slide 2 onward), **draw** the computation graph for **only the first term** of $\mathcal{L}$ for only one training example, i.e., $-y_i \log\left(h\left(\mathbf{w}^\top \mathbf{x}_i + b\right)\right)$.

(Remember: The goal of building a computational graph is to break down a calculation into primitive computing steps such that some of the nodes in the graph would carry the derivatives of the loss function w.r.t. the parameters w_1, w_2, and b. The end goal is to avoid having to compute every single partial derivative from scratch, which is very expensive, especially for complicated models such as neural networks, which can have tens of millions of parameters!)

Note: you can draw the graph however you like, as long as it's readable. It's OK if you draw it on paper and include a clear photo of it. It's OK if you scan it. It's OK if you draw it using Google Drawings or TikZ, etc.

2.4. (2 points) Recall, backpropagation is a fancy name for "chain rule for differentiation with memoization." Fill in the memoized derivatives in each node you drew for the previous exercise, using the same technique as Slide 25. Show both the forward and the backward terms (the red and blue terms in the examples). Clearly label your computation graph with what flows between the nodes in **both directions**.

3 Extra Credit: Numerical Evaluation (3 points)

3.1. (1.5 points) Suppose the values of your parameters are

$$\mathbf{w} = \begin{bmatrix} 1.1 \\ -6.0 \end{bmatrix}, \quad b = 2 \tag{6}$$

and that you have a training example $\mathbf{x} = \begin{bmatrix} 5 & 10 \end{bmatrix}^\top$ (i.e., $x_1 = 5$ and $x_2 = 10$), whose label is $y = 1$.

Using the expressions you derived in Exercise 2.1, compute the values for

$$\frac{\partial \mathcal{L}}{\partial w_1}, \quad \frac{\partial \mathcal{L}}{\partial w_2}, \quad \text{and} \quad \frac{\partial \mathcal{L}}{\partial b} \in \mathbb{R}. \tag{7}$$

3.2. (1.5 points) Write code to implement the computation graph in Exercise 2.3. Fill in the terms numerically using the same values as in 3.1, i.e.,

$$\mathbf{x} = \begin{bmatrix} 5 \\ 10 \end{bmatrix}, \quad y = 1, \quad \mathbf{w} = \begin{bmatrix} 1.1 \\ -6.0 \end{bmatrix}, \quad b = 2, \tag{8}$$

and demonstrate that you get the same values for the derivatives as in Exercise 3.1. You can use any programming language. MATLAB and Python are recommended.