[420] Assignment 5

Tanuj Kumar ta.kumar@mail.utoronto.ca 1002197133

December 2017

1 Tracking

1.1 Greedy Algorithm

1.1.1 Explanation of the Algorithm

The algorithm stores a **track set** as a two-dimensional linked-list (Python list) structure, a list of **tracks**. For each image iteration:

- Generate the similarity matrix of the current image i and the next image j.
- Retrieve the coordinates (k, t) of the element in the similarity matrix with the maximum value at the moment. Form tuples (i, k) and (j, t).
- For each current track in the track set, check if the last element in the track is a tuple of the form (i, x), where x is some matrix index from the last iteration. That is, i must match (i, k). If this is true, then simply add (j, t) to that track.
- If no such track is found, then create a new track with (i, k) and (j, t).
- In the matrix, zero out all elements in row k and column t. This allows for all track tuples to be disjoint.
- Repeat to find the next maximum, until all elements in the matrix are 0. Move on to the next image.

An example of a track: $(62, 1), (63, 4), (64, 7), (65, 0)$ which is a track that spans images 62 to 65 and includes detections with indices 1, 4, 7, and 0 as the "same" object moving among the frames.

From the above, tracks are guaranteed to have disjoint elements and no repeats, along with at most one frame detection per image per track. The algorithm then simply searches the track set for all tracks greater than 2 elements in length, and adds these to the final list (or conversely, removes all tracks less than or equal to 2 elements in length).

1.1.2 Code

track-objects.py (only the helper function "visualize players" and the main algorithm are given below, as all other code in the document is just given):

```
def visualize_players(tracks):
  """Visualizes the detected moving players.

  Args:
    tracks:      A 2D list consisting of tuple-lists, representing
        individual tracks.

  Returns:
    (none)       uses showboxes.py for visualization.
  """

  culled_tracks = [tracks[k] for k in range(len(tracks)) if
      len(tracks[k]) > 5]

  color = ['r', 'c', 'b', 'y', 'm', 'navy', 'turquoise', 'darkorange',
      'cornflowerblue']

  valid_colors = color[0:len(culled_tracks)]

  for image_id in range(start_frame, end_frame):

    current_image, current_detections =
        load_image_and_detections(image_id)

    detection_indices = []
    track_colours = []

    for track in range(len(culled_tracks)):

      current_track = culled_tracks[track]
      image_detections = [current_track[det] for det in
          range(len(current_track)) if current_track[det][0] == image_id]
      if (len(image_detections) == 1):
        detection_indices.append(image_detections[0][1])
        track_colours.append(valid_colors[track])

    detection_coordinates = []

    for ind in detection_indices:
      detection_coordinates.append(current_detections[ind,0:4])

    savepath = 'C:/<private path
        content>/csc420/a5/Assignment5/tracks/'+str(image_id)+'.png'

    showboxes(current_image, detection_coordinates, track_colours,
        savepath)


start_frame = 62
end_frame = 72

sims = []
tracks = []

for frame_id in range(start_frame, end_frame):
  current_image, current_detections = load_image_and_detections(frame_id)

  next_image, next_detections = load_image_and_detections(frame_id + 1)

  # sim has as many rows as len(current_detections) and as many columns
      as
  # len(next_detections).
  # sim[k, t] is the similarity between detection k in frame i, and
      detection
  # t in frame j.
  # sim[k, t] == 0 indicates that k and t should probably not be the
      same track.
  sim = compute_similarity(current_detections, next_detections,
                          current_image, next_image)

  sim_copy = sim
  # refNull = np.zeros(simCopy.shape)
  # isNull = not refNull.any()

  for i in range(sim_copy.shape[0] * sim_copy.shape[1]):

    k, t = np.unravel_index(sim_copy.argmax(), sim_copy.shape)
    if (sim_copy[k,t] == 0):
      break # no more maxes
    frame_id_tuple = (frame_id, k)
    next_id_tuple = (frame_id+1, t)
    new_track = [frame_id_tuple, next_id_tuple]

    num_tracks = len(tracks)
    added = 0

    for j in range(num_tracks):
      track_length = len(tracks[j])
      last_track_item = tracks[j][track_length-1]
      if frame_id_tuple == last_track_item:
        tracks[j].append(next_id_tuple)
        added = 1

    if added == 0:
      tracks.append(new_track)
    sim_copy[k,:] = 0
    sim_copy[:,t] = 0

visualize_players(tracks)
```

showboxes.py:

```
"""Python function for showboxes()

If no plot shows up after calling this function, try:

>>> import pylab
>>> showboxes(...)
>>> pylab.show()
"""

import matplotlib.patches as patches
import matplotlib.pyplot as plt
import numpy as np
# import pylab

def showboxes(image, boxes, colors, output_figure_path=None):
  """Draw bounding boxes on top of an image.

  Args:
    image:              PIL.Image object
    boxes:              A N * 4 matrix for box coordinate.
    output_figure_path: String or None. The figure will be saved to
                        output_figure_path if not None.
  """
  figure = plt.figure()
  axis = figure.add_subplot(111, aspect='equal')
  colorind = 0
  for i,box in enumerate(boxes):
    axis.add_patch(patches.Rectangle(box[:2],
                                     box[2] - box[0],
                                     box[3] - box[1],
                                     fill=None,
                                     ec=colors[i],
                                     lw=3))

  plt.imshow(image)
  # pylab.show()

  if output_figure_path is not None:
    plt.savefig(output_figure_path)
```

1.2 Solution Visualization

See the attached .zip folder. Visualization was achieved by assigning each track a colour, and then simply drawing the boxes corresponding to each specific image for each track.

1.3 Missing Detections

There are a number of ways to deal with missing detections. If limited by the existing architecture and processes at hand, then one key idea would be to look for "holes" in disparity paths. Namely, situations where sequences of images in tracks appear to go on as normal, but then there is a sudden drop before continuing, such as $(61, 1), (62, 4), (64, 6)$ missing image 63. With the normal greedy algorithm above, this would split up the above track into two separate tracks, but if the detection in image 63 can be found to "stitch" together the track components to create a full track, this will easily replace a missing detection. A quick way this can be done is, after the greedy algorithm, iterate through tracks, looking at the last image IDs and first image IDs in each track. If there are any matches between last and first that differ by a threshold number of images (for example, one track ends with frame 63 while another track starts with frame 65), they *may* be a candidate for a stitching together. However, this does not guarantee that these objects are the same. The actual similarity matrix and the DPM data should be investigated.

On top of the above, two general methods to fix missing detections include a more precise DPM setup that can more accurately capture fine player motion from this camera distance as detection data, or simply using something like a CNN which has a high accuracy of player detection off the bat as the basis of presenting reliable detection data. In both cases, this helps the capturing of players that otherwise were not captured by this initial DPM implementation, and can be more finely tuned to deal with situations where detection can be a bit more complicated, such as when players stand in front of each other in the camera and obscure views (this is also a possible cause for "holes", as explained above).

1.4 Calculating Speed

If working explicitly from the given material and information, the best way to calculate speed would first involve a homography transformation on the *soccer field* to get an accurate displacement estimate of each player, based on the recording of their movements when compared between frames on their tracked paths. The soccer field homography will help transform the movements of the players into the context of the soccer field's distances, and based on standardized measures of soccer field dimensions we can measure the actual distances, and finally use the resulting displacement data with frame capture time to calculate player velocity.

If more camera information is provided, such as a stereo view (via a second stadium camera), we can calculate a depth map for the soccer field area and the players.

Using this depth map, we can transform image coordinates into world coordinates, and then get the player displacements in world coordinate data, allowing us to then use the time between frames to calculate velocity in world coordinates.

2 Computation Graphs and Deep Learning

2.1 Analytical Derivative Expressions

Using a few tricks to simplify the equation, we can calculate all three partial derivatives at once. For a point $(y_i, \mathbf{x}^i)$ and starting with the assumption that $i \to M$, we have:

$$L(\mathbf{w}, b) = -\frac{1}{M}\sum_{i=1}^{M}[y_i \log(h(\mathbf{w}^T\mathbf{x}^i + b)) + (1 - y_i)\log(1 - h(\mathbf{w}^T\mathbf{x}^i + b))$$

Where:

$$h(\mathbf{w}^T\mathbf{x} + b) = \frac{1}{1 + e^{-\mathbf{w}^T\mathbf{x} - b}}$$

We can simplify the log terms as follows, via laws of logarithms:

$$\log(h(\mathbf{w}^T\mathbf{x}^i + b)) = -\log(1 + e^{-\mathbf{w}^T\mathbf{x} - b})$$

$$\log(1 - h(\mathbf{w}^T\mathbf{x}^i + b)) = -\mathbf{w}^T\mathbf{x} - b - \log(1 + e^{-\mathbf{w}^T\mathbf{x} - b})$$

We can then rewrite the loss equation:

$$L(\mathbf{w}, b) = -\frac{1}{M}\sum_{i=1}^{M}[-y_i \log(1+e^{-\mathbf{w}^T\mathbf{x} - b}) + (1-y_i)(-\mathbf{w}^T\mathbf{x} - b - \log(1+e^{-\mathbf{w}^T\mathbf{x} - b}))]$$

Simplifying and cancelling like terms, we get:

$$L(\mathbf{w}, b) = -\frac{1}{M}\sum_{i=1}^{M}[y_i(\mathbf{w}^T\mathbf{x} + b) - (\mathbf{w}^T\mathbf{x} + b) - \log(1 + e^{-\mathbf{w}^T\mathbf{x} - b})]$$

$$L(\mathbf{w}, b) = -\frac{1}{M}\sum_{i=1}^{M}[y_i(\mathbf{w}^T\mathbf{x} + b) - (\log(e^{\mathbf{w}^T\mathbf{x} + b}) + \log(1 + e^{-\mathbf{w}^T\mathbf{x} - b}))]$$

$$L(\mathbf{w}, b) = -\frac{1}{M}\sum_{i=1}^{M}[y_i(\mathbf{w}^T\mathbf{x} + b) - \log(1 + e^{\mathbf{w}^T\mathbf{x} + b})]$$

From here, since $i = 2$, we can expand out the inner product:

$$L(\mathbf{w}, b) = -\frac{1}{M}\sum_{i=1}^{M}[y_i(w_1x_1^i + w_2x_2^i + b) - \log(1 + e^{w_1x_1^i + w_2x_2^i + b})]$$

Taking partials of these sub-terms:

$$\frac{\partial}{\partial w_j}(y_i(w_1x_1^i + w_2x_2^i + b)) = y_ix_j^i$$

$$\frac{\partial}{\partial w_j}(\log(1 + e^{w_1x_1^i + w_2x_2^i + b})) = x_j^ih(\mathbf{w}^T\mathbf{x} + b)$$

And we treat $b = w_0$ and $x_0 = 1$, then $\frac{\partial}{\partial b} = \frac{\partial}{\partial w_0}$ in the above situations.

Therefore, the partial derivatives are:

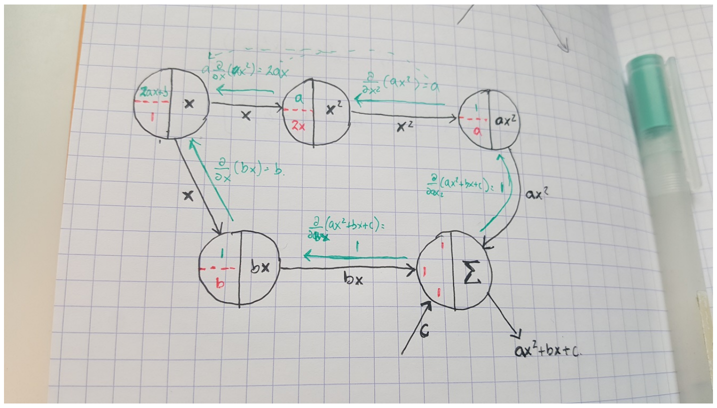
$$\frac{\partial L}{\partial w_1} = -\frac{1}{M}\sum_{i=1}^{M}(y_ix_1^i - x_1^ih(\mathbf{w}^T\mathbf{x} + b))$$

$$\frac{\partial L}{\partial w_2} = -\frac{1}{M}\sum_{i=1}^{M}(y_ix_2^i - x_2^ih(\mathbf{w}^T\mathbf{x} + b))$$

$$\frac{\partial L}{\partial b} = -\frac{1}{M}\sum_{i=1}^{M}(y_i - h(\mathbf{w}^T\mathbf{x} + b))$$

2.2 Warm-up Computation Graph

Below is the warm-up computation graph:



$a\frac{\partial}{\partial x}(ax^2) = 2ax$
$\frac{\partial}{\partial x^2}(ax^2) = a$
$2ax+b$
1
x
x
a
2x
x^2
x^2
1
a
ax^2
$\frac{\partial}{\partial x}(bx) = b.$
x
$\frac{\partial}{\partial ax_2}(ax^2+bx+c) = 1$
ax^2
$\frac{\partial}{\partial bx}(ax^2+bx+c) =$
1
1
b
bx
bx
1
1
1
Σ
c
ax^2+bx+c

Figure 1: Warm-up C-Graph

2.3 Full Computation Graph (2.3 and 2.4)

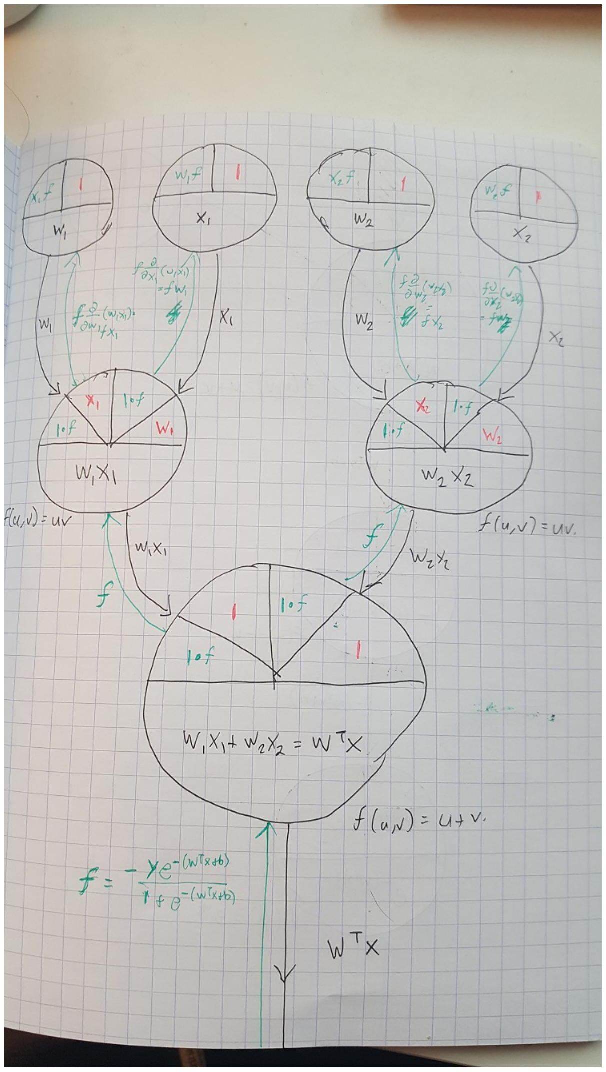


Figure 2: C-Graph Part 1

3 Bonus

Let $w_1 = 1.1, w_2 = -6.0, b = 2, x_1 = 5, x_2 = 10, y = 1$. Then:

$$w_1x_1 + w_2x_2 + b = (1.1)(5) + (-6.0)(10) + (2) = -52.5$$

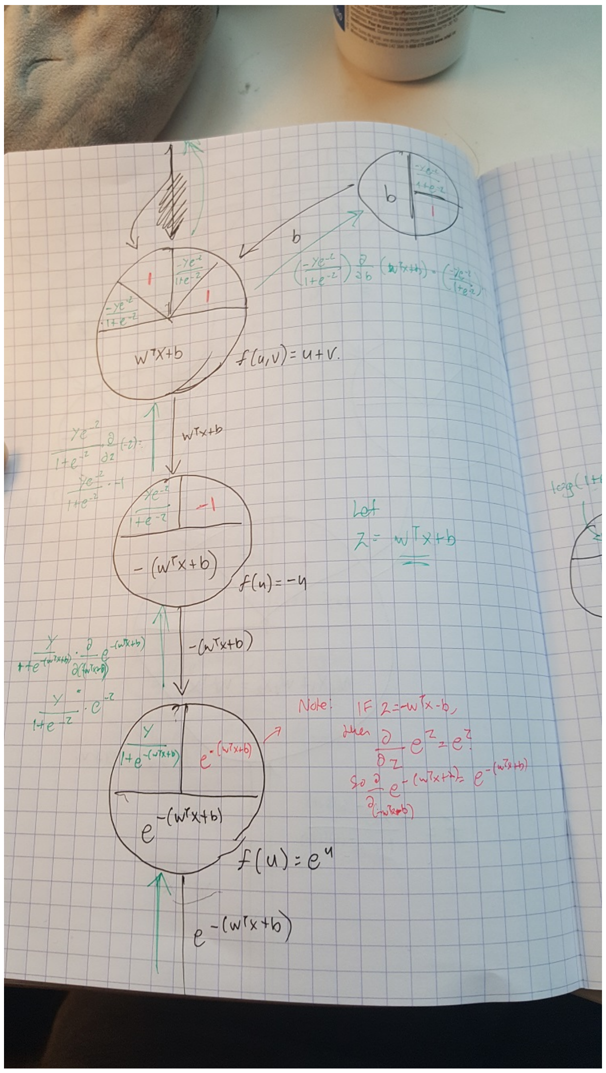


Figure 3: C-Graph Part 2

$$h(-52.5) = 1.583 \times 10^{-23}$$

$$\frac{\partial L}{\partial w_1} = -((1)(5) - (5)(1.583 \times 10^{-23}) = -4.99$$

$$\frac{\partial L}{\partial w_1} = -((1)(10) - (10)(1.583 \times 10^{-23}) = -9.99$$

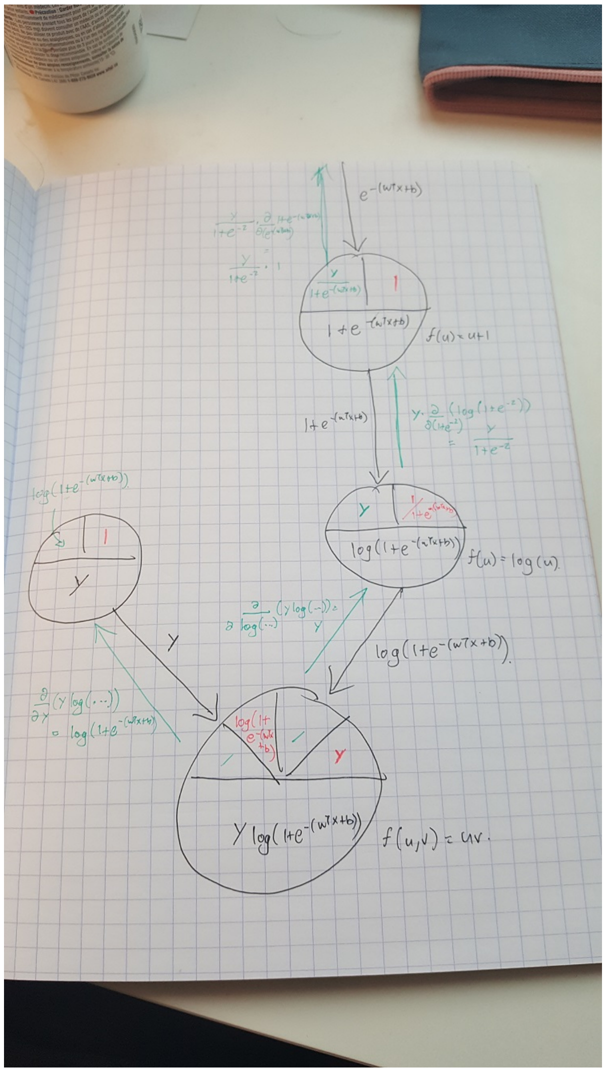

$e^{-(w^Tx+b)}$
$\frac{y}{1+e^{-z}} \cdot \frac{\partial}{\partial(e^{-(w^Tx+b)})} 1+e^{-(w^Tx+b)}$
$= \frac{y}{1+e^{-z}} \cdot 1$
$\frac{y}{1+e^{-(w^Tx+b)}}$
1
$1+e^{-(w^Tx+b)}$
$f(u) = u+1$
$1+e^{-(w^Tx+b)}$
$y \cdot \frac{\partial}{\partial(1+e^{-z})}(\log(1+e^{-z})) = \frac{y}{1+e^{-z}}$
$\log(1+e^{-(w^Tx+b)})$
y
$\frac{1}{1+e^{-(w^Tx+b)}}$
$\log(1+e^{-(w^Tx+b)})$
$f(u) = \log(u)$
1
y
y
$\frac{\partial}{\partial \log(\ldots)}(y\log(\ldots)) = y$
$\log(1+e^{-(w^Tx+b)})$
$\frac{\partial}{\partial y}(y\log(\ldots)) = \log(1+e^{-(w^Tx+b)})$
$\log(1+e^{-(w^Tx+b)})$
y
$y\log(1+e^{-(w^Tx+b)})$
$f(u,v) = uv$

Figure 4: C-Graph Part 3

$$\frac{\partial L}{\partial b} = -((1) - (1.583 \times 10^{-23}) = -0.99$$