MTRE 4490 – Machine Learning for Robot Perception
Project – Cone Detection and Position Estimation

Project goal

This is the first course project to handle regression, i.e. the output of the machine learning algorithm is a continuously changing variable rather than a classification result. The International Aerial Robotics Competition involves "herding" Roombas by landing on them with a drone to trigger motion in a desired direction. One challenge with this task is identifying which direction the Roomba is facing, as illustrated in Figure 1. This regression problem will be solved using a backpropagation artificial neural network (ANN).

Figure 1: Roomba with its orientation marked with the red arrow.

Project submission guidelines

Create a folder named LastNameProject04 containing all necessary source code files, image files for output similar to Figures 2 and 3, and the artificially generated training/validation images. Place the images in their own folder where image file names consist of xxxx_y.jpg where xxxx represents any string without underscores or periods, and y represents an integer for the rotation angle of the image. Source code files should include one that loads training data, produces figures similar to Figure 2 and 3, and saves the best ANN to a file. Also include LastNameProject04.py which loads the saved ANN and generates figures similar to Figure 4 and 5 by loading the 360 images created from the Roomba02.jpg file. This code should be written so that the file path can simply be altered to load all images with names test_x.jpg when evaluated by the instructor.

Training and validation data

Supplied with this project are two files Roomba01.jpg and Roomba02.jpg. Although resolution in each image is different, the Roomba is located at the center with a size of 200×200 pixels in both images. In both cases, the Roomba is pointing to the right, i.e. at an angle of 0°. Create artificial data by rotating the images at increments of 1°, saving the image with a filename ending with _y where y is the rotation degrees. Doing so enables building the labels for each image automatically based on file name. For example, the code

```
import glob
fname = glob.glob('images/train/*.jpg')
for i in range(len(fname)):
    beg = fname[i].index('_')
    end = fname[i].index('.')
    angle = int(fname[i][beg+1:end])
    print(angle,type(angle))
```

displays the angle, as an integer, for each of the properly named images in the indicated folder. From the two supplied images, 720 samples are now available for training/validating the ANN. Similar to the cone classification problem, build a feature matrix using HOG and color histogram features. Consider using a `sklearn.preprocessing.StandardScaler` on the feature matrix, although this may or may not improve

performance. Split the data into 80% training and 20% validation data for early stopping using `sklearn.model_selection.train_test_split`.

Sample labels

The desired output of the ANN is the direction the Roomba is pointing. A single ANN output of the angle is problematic since feature inputs for angles of 0° and 359° are similar and yet require significantly different output values. Instead, an output encoding the periodicity of direction is more effective. To achieve this, the ANN could be trained with the sine (or cosine) of the angle rather than the angle itself since $\sin(x) = \sin(x+360°)$. The predicted angle would then be determined by computing the inverse sine of the ANN output. A problem with this approach is that inverse sine is only defined on a domain of [-1,1] which the ANN is not guaranteed to respect. Additionally, the sine function is not one-to-one: every sine output value has two possible input x values. Instead, the ANN could be trained with two labels, both the sine and cosine of the ground truth angle. The two ANN outputs y and x would predict the angles using

$$\theta = \tan^{-1}\left(\frac{\sin\theta}{\cos\theta}\right) \approx \tan^{-1}\left(\frac{y}{x}\right)$$

which is always defined even for predicted values of y and x not on the domain [-1,1] as shown in Figure 1 (left). Using the standard `np.atan` function introduces a different problem in that the 180° periodicity of tangent only returns angles between -90° and 90°. For example

$$\tan^{-1}\left(\frac{1}{1}\right) = \tan^{-1}\left(\frac{-1}{-1}\right) = \tan^{-1}(1) = 45°$$

even though the first example actually represents 45° but the second 225°. The four-quadrant `np.atan2` overcomes this problem by taking both numerator and denominator as independent input arguments, therefore `np.atan2(1,1)` can be distinguished from `np.atan2(-1,-1)`. Figure 1 (right) illustrates the four quadrant inverse tangent resulting in values ranging from -180° to 180°. Note that predictions of angles less than zero should have 360° added to them when comparing with ground truth labels.

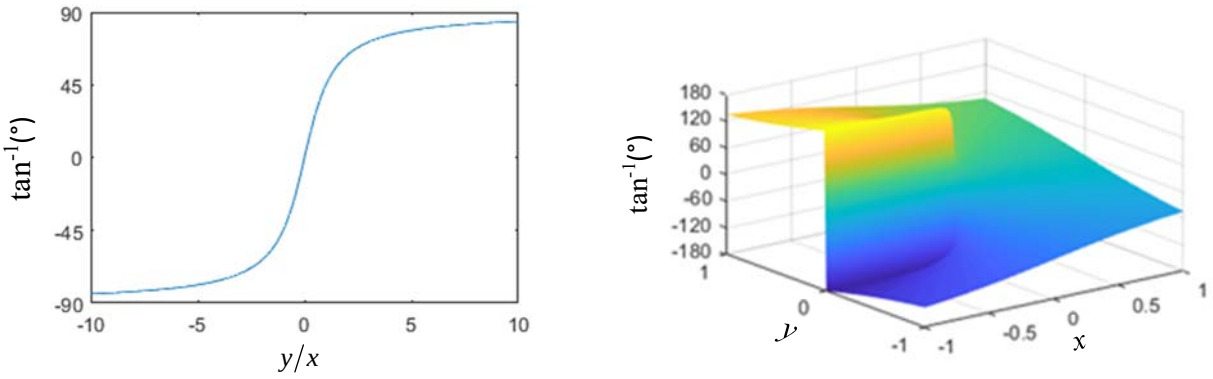

Figure 1: Inverse tangent using the standard `np.atan` (left) and the four quadrant `np.atan2` (right).

Labels for the resulting two-output problem should be organized not as a 1D array, but rather a 2D array with two columns and a row for each sample image.

ANN training

Early stopping

The 20% of data set aside for validation should be used to prevent overfitting by stopping training when performance in the validation set fails to decrease, even if training performance continues to improve. Figure 2 depicts a typical

training process where training and validation performance begin approximately the same but eventually separate as the ANN optimizes explicitly on the training set. Training was halted at the right end of Figure 2 when validation performance failed to decrease over a certain number of epochs. Notice the logarithmic vertical axis in Figure 2, generated with `matplotlib.pyplot.semilogy`, allows discrimination between training and validation set performance even as errors become increasingly small. The mean absolute errors (MAEs) reported are calculated with

$$\text{MAE} = \sum_{i=1}^{n} \left| \sin\theta_i - y_i \right| + \sum_{i=1}^{n} \left| \cos\theta_i - x_i \right|$$

where n is the number of samples, θ_i is the ground truth angle for sample i, and the ANN output for the i^{th} sample is

$$\mathbf{y}_i = \begin{bmatrix} y_i \\ x_i \end{bmatrix}$$

with y_i and x_i predicting $\sin(\theta_i)$ and $\cos(\theta_i)$, respectively. While not expressed as an angle, this MAE represents the error function the ANN is directly working to reduce.

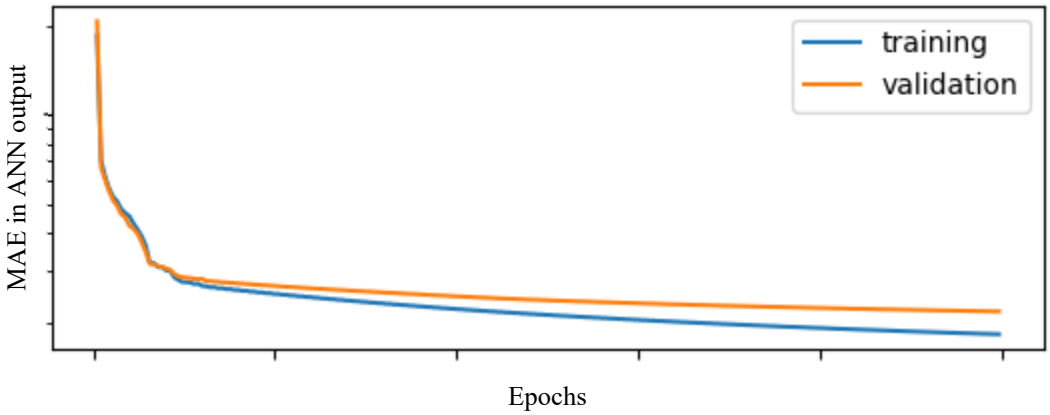

Figure 2: Drop in mean absolute error of ANN outputs (scaled logarithmically) for both training and validation data for a typical optimization run.

Hidden layer size

Determining the proper number of hidden nodes for the ANN is complicated with the fact that training results are dependent on initial weight values. Comparing performance of a single run at various hidden layer sizes is unlikely to provide clear direction on the optimal size. Instead, a number of runs should be performed for each layer size and the average of each compared as in Figure 3, which illustrates typical behavior where too few hidden nodes results in weaker performance. In theory, adding more hidden nodes should never result in weaker performance since weights for the additional nodes could be set to zero, resulting in the exact same smaller network. However, the error surface for larger networks is more complicated and training can become trapped in the increased number of local minima. For this reason, performance typically degrades for excessively increased numbers of nodes as shown in Figure 3, accompanied with rising computation time as well. The MAE in Figure 3 is expressed in degrees, using

$$\text{MAE}_{\text{deg}} = \sum_{i=1}^{n} \left| \theta_i - \tan^{-1}\left(y_i, x_i \right) \right|$$

Note that selecting the number of hidden nodes is more appropriately based on validation as opposed to training set performance because it offers a better indication of performance on data not involved in the training process. Using data from a similarly created Figure 3, select and save the ANN with the best performance to later be used for evaluating test set images.

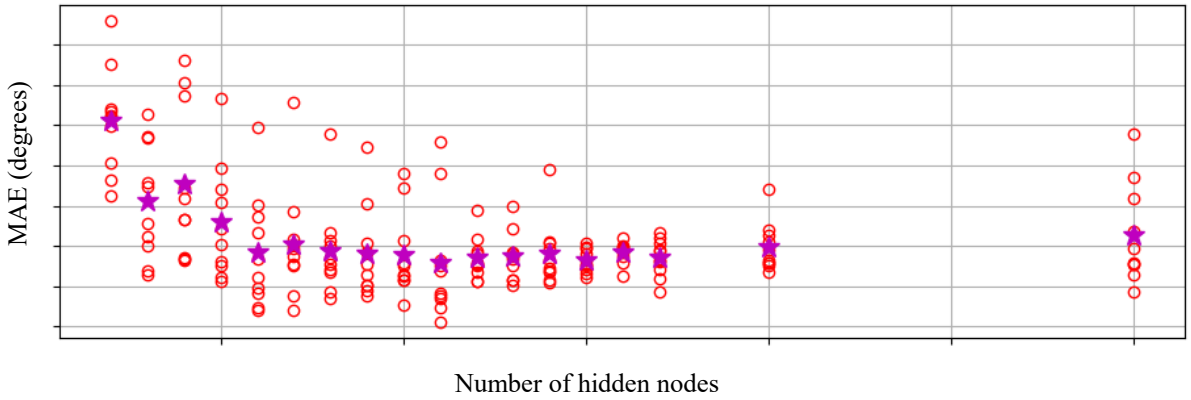

Figure 3: Variation of mean absolute error (in degrees) for validation set data with respect to number of hidden nodes. Individual runs and average values marked with circles and stars, respectively.

Test set evaluation

Once trained, the selected best ANN is then applied to making predictions for unknown test data. This project will be partially evaluated on its performance on test images, one for each integer angle from 0 to 359°, derived from a different image than the two supplied for training. Figure 4 displays the error in the predicted angle for each image in the test set. Additionally, display on the screen the MAE in degrees for the entire test set and the maximum absolute error in degrees, i.e. error for the worst prediction. Without access to test set images, write code to load images from Roomba02.jpg, and the instructor will change the path to evaluate test images.

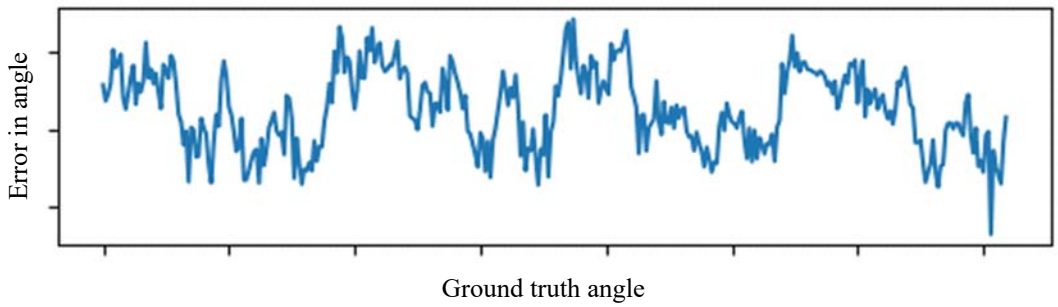

Figure 4: Performance in test set with best ANN selected during training process.

As a final visual representation of ANN performance, display each of the 360 test images with an arrow drawn from the center indicating predicted direction. As an example, the code

```
import matplotlib.pyplot as plot
fig = plot.figure(1)
plot.clf()
ax = fig.gca()
#        center  dx   dy
ax.arrow( 10,5,  20, -30, fc="c",ec='m',head_width=5,head_length=15,linewidth=3)
plot.axis('equal')
plot.xlim(  0, 80)
plot.ylim(-40, 10)
```

draws an arrow with cyan lines and magenta fill centered at (10,5) with total length of

$$\sqrt{20^2+30^2}+5+15=56$$

at an angle of

$$\theta = \tan^{-1}(-30, 20) = -56.3°$$

Use `matplotlib.pyplot.subplot(15,24,i)` to create the matrix of 360 images in Figure 5 including arrows for directions. Images will be small but should be legible when maximizing the figure to full screen.

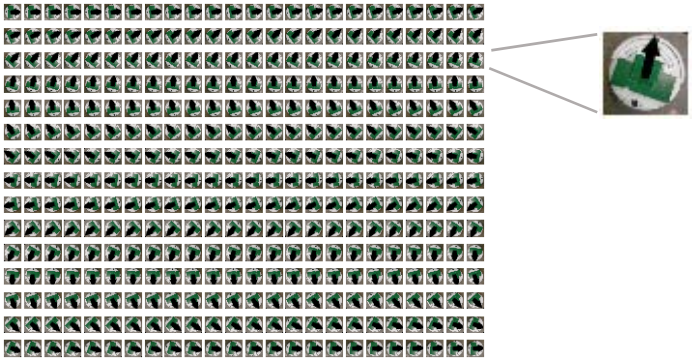

Figure 5: Visual depiction of test set performance in a 15×24 subplot with an arrow for the predicted direction drawn over each image.

Grading rubric

- (10 points) Training images created for all angles from images Roomba01.jpg and Roomba02.jpg
- (10 points) HOG and color histogram feature matrix created for training images
- (5 points) Data split into training and validation sets
- (10 points) Labels of $\sin(\theta_i)$ and $\cos(\theta_i)$ used to train two output ANN
- (10 points) Train ANN neural network using early stopping
- (10 points) Multiple runs with different starting weights executed for each network size
- (5 points) Best ANN selected after testing appropriate range for number of hidden nodes
- (5 points) Submitted image file similar to Figure 2
- (5 points) Submitted image file similar to Figure 3
- (5 points) Code generates figure similar to Figure 4
- (5 points) Code generates figure similar to Figure 5
- (10 points) Roomba02.jpg performance: MAE under 1.5° and maximum absolute error under 5°
- (10 points) Test image performance: MAE under 5° and maximum absolute error under 15°