**User Guide for Adversarial Pedestrian Control**

The purpose of this research is to teach a simulated pedestrian to get itself hit by an autonomous vehicle. The value of being able to do that is that you can find flaws in your autonomous vehicle software before you release it.

**Step 1: Generating Baseline Data**

Before the pedestrian can be trained, it needs data with which to be trained. The file *randomly\_controlled\_pedestrian.jl* accomplishes this.

Edit the *numDrives* variable to reflect how many drives you want to simulate.

Edit the *printDrive()* function to tell it what path to save the data it produces.

Run the command *driveTracking* = *basicExecute(driveTracking)* to simulate the drives. This will simulate a specified number of drives in which the pedestrian acts randomly.

Run the command *printDrive(driveTracking)* to save the data to CSVs. Three CSVs are made. One has all the data in one place for reference. The other two split the data into inputs (state & action) and labels (cost).

Do Step 1 twice, once to create training data and once to create testing data.

**Step 2: Training the Initial Neural Net**

Now an initial neural net can be trained. This can be done inside either *deep\_neural\_net.py* or *two\_layer\_neural\_net.py* with the same instructions for each.

Change the paths where *inputs\_test, labels\_test, inputs\_train* and *labels\_train* look for their information to correspond to where you saved you CSVs in part 2.

Make sure *reuse* is set to False.

Change the paths where the weights and biases are saved to at the end to be where you want them.

Run *two\_layer\_neural\_net.py* or *deep\_neural\_net.py* to train the model and generate weights and biases.

**Step 3: Using the Neural Net to Control the Pedestrian**

Now we can use the model we trained in step 2 to control the pedestrian and generate new data based on the new pedestrian.

For this step, you will use *net\_controlled\_pedestrain.jl* in a similar manner as *randomly\_controlled\_pedestrian.jl*.

Edit the *numDrives* variable to reflect how many drives you want to simulate.

Edit W1\_ - W3\_ and b1\_ - b\_3 to reflect the locations where weights and biases were saved in Step 2.

Edit the *printDrive()* function to tell it what path to save the data it produces.

If using *deep\_neural\_net.py* to generate model, changes will need to be made to the weights and biases in *net\_controlled\_pedestrain.jl* so that it accepts the right shaped inputs. It is currently set up to only work for *two\_layer\_neural\_net.py.*

Run the command *driveTracking* = *basicExecute(driveTracking)* to simulate the drives. This will simulate a specified number of drives in which the pedestrian acts according to the neural net. At each timestep, the pedestrian will look at all of its available modes, determine which ones it thinks will be associated with the lowest cost, and pick a move accordingly.

Run the command *printDrive(driveTracking)* to save the data to CSVs. Three CSVs are made. One has all the data in one place for reference. The other two split the data into inputs (state & action) and labels (cost).

Do Step 3 twice, once to create training data and once to create testing data.

**Step 4: Training Subsequent Neural Nets**

Now subsequent neural nets can be trained iteratively. This can be done inside either *deep\_neural\_net.py* or *two\_layer\_neural\_net.py* with the same instructions for each.

This is the same as Step 2 except *reuse* should be set to true and the paths for finding weights and biases should be checked to make sure they are looking in the right place.

**Step 5: Iterate**

Steps 3 and 4 can be repeated as needed. The more times this process is iterated, the more the neural net can hone in on areas of interest and get better at discriminating between more important choices.

Note: There is a currently unresolved bug when passing weights between the many files that interpret and produce them where they come in as the transpose of what they should be. If you ever run into this, just transpose the offending party so that the matrix multiplication works out and it should be fine.