Conor Sayres
Astro 598
Fall 2019
Final Project

Goal: Train a neural network to efficiently associate targets (specified in x,y coordinates) to SDSS-V robotic fiber positioners under collisional constraints. Tl;dr, didn't work so well.

Project Repo: https://github.com/csayres/astro598

Repo contents:
-Main script to run analysis: targAssign.py
-Figures generated: *.png
-Trained and saved keras model: targAssign.h5
-Anticollision.pdf: background for geometries involved (you've already seen)
-This writeup.
-[Not included]: validAssign_*.p, a pickle file containing valid target associations (too big to include). This can be generated using the generateAssignments function in the main script.

Assigning targets to fibers is a tricky problem to optimize in the context of the SDSS-V focal plane system because robots interfere spatially (see anticollision.pdf for a reminder of how robots collide). The kaiju package (https://github.com/sdss/kaiju) is being actively developed to solve the anti-collision robot navigation problem for SDSS-V, and it can be used to generate massive example sets of non-colliding target orientations. I'm interested in investigating the feasibility of training a neural network to pick good, if not optimal, robot assignments for a set of targets.

The neural network I designed is a multi-layer fully connected NN with a single hidden layer. I chose to use a hexagonal grid containing 37 positioners. Each robot gets a target specified by [x,y], so the input layer requires 37x2 = 74 nodes. The hidden layer has 111 nodes. The output layer is massive: 37 targets x 37 positioners = 1369 nodes. Using kaiju, I created a set of 1000000 valid target assignments. 90% of these were used for training, and the remainder for verification. The keras output from the fitting is below:

Model: "sequential_1"

Layer (type)	Output Shape	Param #
dense_1 (Dense)	(None, 111)	8325
dense_2 (Dense)	(None, 1369)	153328

Total params: 161,653

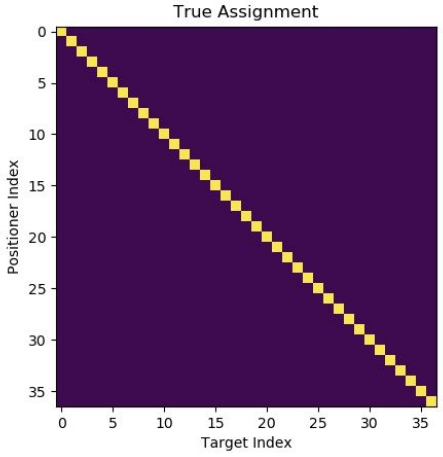
Trainable params: 161,653
Non-trainable params: 0

_____

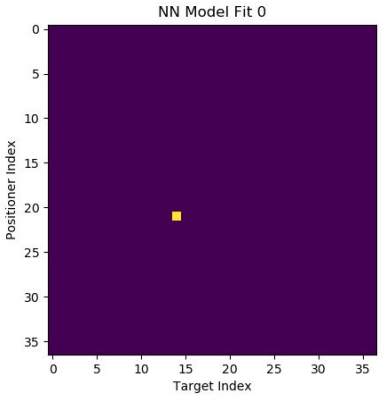
Train on 900000 samples, validate on 100000 samples
Epoch 1/5
900000/900000 [==============================] - 105s 116us/step - loss: 55634.9269 - accuracy: 0.0132 - val_loss: 155071.0203 - val_accuracy: 0.0151
Epoch 2/5
900000/900000 [==============================] - 101s 113us/step - loss: 347829.1262 - accuracy: 0.0158 - val_loss: 585698.9901 - val_accuracy: 0.0159
Epoch 3/5
900000/900000 [==============================] - 101s 112us/step - loss: 917789.4078 - accuracy: 0.0162 - val_loss: 1296174.2329 - val_accuracy: 0.0163
Epoch 4/5
900000/900000 [==============================] - 100s 112us/step - loss: 1766954.5764 - accuracy: 0.0162 - val_loss: 2286300.3043 - val_accuracy: 0.0161
Epoch 5/5
900000/900000 [==============================] - 100s 112us/step - loss: 2895775.1287 - accuracy: 0.0164 - val_loss: 3554400.5307 - val_accuracy: 0.0162

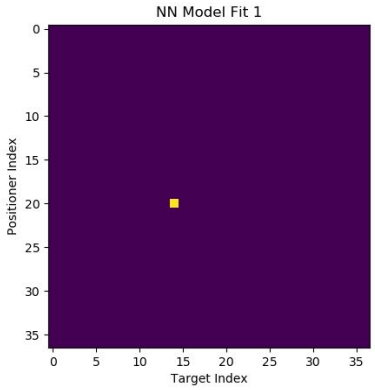
Despite all that training, a whopping 0.0162 accuracy was achieved (if you think that's bad you should see my initial attempts)! So things don't look so promising but below are some visualizations of the results.

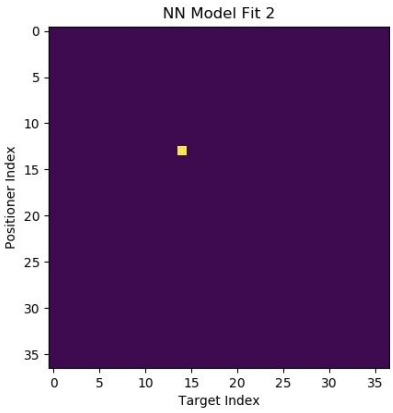
A correctly assigned array should contain one target for one robot in the output robot x target matrix. The figure below shows this for the case that target1->robot1, target2->robot2, … targetN -> robotN.

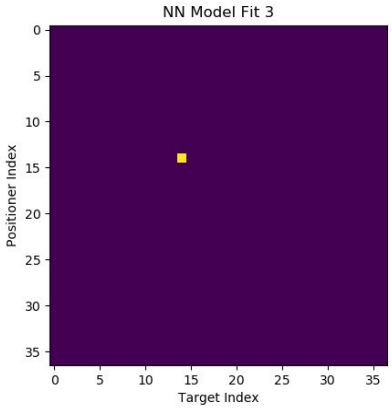


The following visualizations show the types of results the keras fit was giving me:









So basically it was only assigning one target to one robot (small victories). I think that with some more practice and experimentation, a NN could do well at this, but scaling it up from a 37 to a 500 positioner grid will be a hefty amount of additional parameters. Some expert advice would be nice in getting further on this problem. The fact that the output layer is a much higher dimension than the input me feel like 1) I made a grave mistake in the design, or 2) training such a network will require months of crunching training sets, or 3) both.