Homework #1- Machine Learning for Robotics (RBE 577) Control Allocation via Deep Neural Networks

Eric Viscione, Paul Crann
RBE577: Machine Learning For Robotics

Introduction:

Training a neural network to solve the control allocation part of a control system can be useful to have efficient control of a complicated system. In this assignment we look at a method to use a neural network to allocate control inputs to the thrusters of a vessel to keep it steady in port and control its direction during travel. Basing the work off of Skulstad et al, a Multi-layer perceptron encoder-decoder neural network is trained on data generated with a random walk. The data is generated by performing a random walk between the defined boundaries for thruster commands (Forces and angles), and then passing these inputs through the transformation defined in section 3.1 of the paper. The encoder layer takes in the surge, sway and yaw force commands calculated by the systems control system, and outputs the 5 thruster control parameters. These parameters are the thrust demanded for thrusters 1 2 and 3(F1, F2, F3) as well as the angles for the gimbals on thrusters 2 and 3 (A2, A3). During training the resultant vector u = [F1 F2 A2 F3 A3] were fed back into the decoder layer where it generated predicated force command values.

Loss Function:

To generate the loss values for training this model there were 5 loss functions used. The first loss, L0, calculated the mean square error between the input force commands fed into the encoder and the equivalent forces calculated based on the encoder output, u. This loss helps drive the encoder portion of the NN to estimate the transformation from desired input force commands and equivalent thruster commands. The second loss, L1, calculated the mean square error between the input force commands fed into the encoder and the recovered force commands from the decoder layer. This loss helps drive the decoder to learn the opposite relationship as the encoder, mapping thruster commands to desired input commands. The third loss, L2, penalized thruster commands outside of the allowable threshold set for thruster forces and angles. This was implemented in order to force the encoder to return valid thruster inputs. The forth loss, L3, penalized a higher rate of change of thruster forces and angles. This was included to reduce large changes in encoder outputs at each time step, making for smoother control inputs. The fifth loss, L4, aimed to minimize power consumption. It did this by adding a loss for the encoder derived thruster force commands. The last loss, L5, constrained the azimuth ingles of thrusters 2 and 3. This was needed to avoid the decrease in efficiency when the two thrusters are angled opposite of each other, where they would be working against each other. An additional loss was added via L2 regularization in our optimizer, but more on that below.

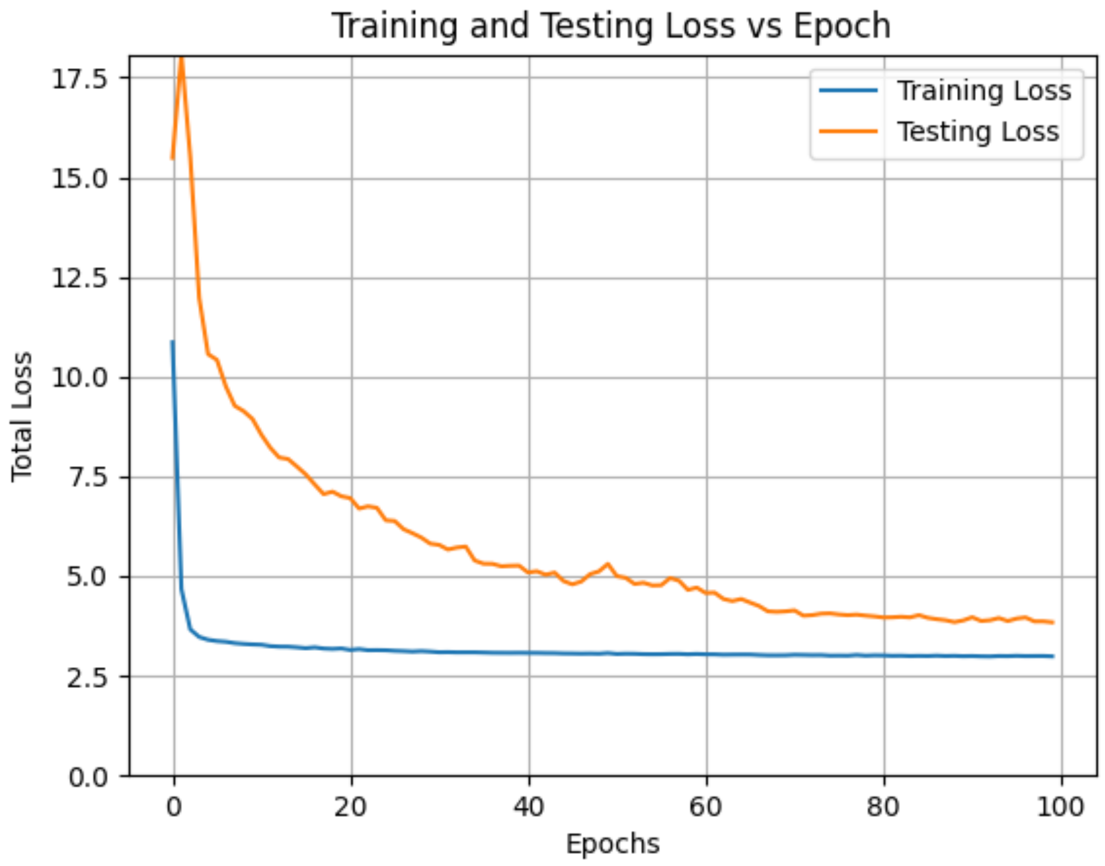
Model Training:

The model used a 3 layer encoder and 3 layer decoder, each with dropout layers, and using ReLU activation functions after the first two layers. Dropout layers were implemented to reduce overfitting. ReLU activation functions were intentionally not used after the final layer of the decoder/encoder due to the strictly positive outputs of the ReLU activation function, and out desired encoder/decoder outputs being evenly distributed above and below 0. The model was optimized using the Adam optimizer with weight decay (L2 Regularization). We used the Adam optimizer because it excels with noisy and sparse data, which is desirable due to our data being generated with a random walk.

Regularization:

Multiple regularization techniques were used in our implementation. Regularization was able to improve our performance across testing data. It accomplishes this by adding small biases to our model in exchange for much decreased variance. This was seen by an increase in training loss after implementing these techniques. The first technique we included was L2 regularization. L2 regularization added a penalty for all weights equivalent to each weight squared. This is intended to reduce the number of very large weights, and therefore, the model is less likely to be reliant on only a few of its weights. The second technique employed dropout layers when training. This randomly sets some weights to zero at each layer, further forcing the model to learn a generalized understanding of the desired function. When inference is done, these are of course turned off.

Results:


Training and Testing Loss vs Epoch
Training Loss
Testing Loss
Total Loss
Epochs