Ryan Masson
Lab Partner: Cary Murray
EECS 301: Intro to Robotics
Professor Argall
12/5/16

Assignment #3: Machine Learning and Smooth Arc Movement

ABSTRACT

In this lab, we learn how the field of machine learning is an important part of modern robotics. Using the machine learning technique locally weighted regression, we build a system that can teach our Bioloid robot to move to positions in its environment in smooth arcs. We tune a parameter of flexibility in our locally weighted regression that can teach us about our dataset. Finally, we achieve fair performance in the learning task, but meet a few challenges, including the inefficiency of the way we collect data and the small size of our dataset.

I. INTRODUCTION

In this class, our robots have gradually gained intelligence, and now we come to the last stop in our journey: machine learning. Machine learning is the ability of a machine to gain new knowledge or abilities through experience, and it has an important place in robotics. From the name and definition of "machine learning", one may think that the systems machine learning researchers build directly rival human intelligence. This is not true, because "they are not meant to be careful models of natural systems, but practical algorithms for helping machines improve behavior."[1]

Implementing a machine learning technique in a robot may be helpful in a number of ways. Machine learning can help a researcher avoid hand-tuning arguments to model-building functions. Robots that learn can build better representations of the environment than the representations that humans can code. When an environment changes frequently —a challenge to any autonomous being— learning can help a robot adapt to the novel environment and still effectively carry out its task. Once one has a reason to use machine learning in a robotic system, different techniques present themselves. In general, "the amount and type of information available to the learning robot determine what type of learning methods are possible for a particular learning problem."[2]

One class of machine learning techniques is known as reward-based learning, where a machine learns by feedback from its environment. Popular within this class is reinforcement learning. In reinforcement learning, a robot tries different actions on its own and observes the results. Generally, depending on the direction/magnitude of the results as assessed by a reward function, the robot receives *reward* if the action did something good and receives *punishment* if the action did something bad. This process results in learning, as the robot learns a relationship between a state transition function and the reward function. As is noted in our textbook, "the most important thing to realize is that the robot uses reinforcement learning to learn state-action combinations."[3] The complete set of combinations is then called the *control policy,* which governs the actions of the robot in response to different states.

The other main class of learning is instance-based learning. In this type of learning a machine compares new situations with previous instances that are explicitly represented as data and used to train the machine. Within instance-based learning are two categories: unsupervised learning and

---

[1] Matarić, p. 260

[2] Matarić, p. 267

[3] Matarić, p. 258

supervised learning. In unsupervised learning, a machine does not have explicit labels on the training data, and must establish meaning on its own. In supervised learning, "the robot gets the full answer in the form of the magnitude and direction of the error."[4] Based on the training data, a supervised learning system uses an algorithm to tune model parameters, all in the service of learning how to do a task. Popular such algorithms include k-nearest neighbors, locally weighted regression, support vector machines, and neural networks.

For all its uses and techniques, machine learning is a difficult endeavor. Learning is messy due to "uncertainty, dynamically changing environments, and hidden/partially observable state."[5] In reinforcement learning, for example, determining what model of feedback to use is difficult, especially when the environment is changing. Also, there are can be drawbacks to machine learning that outweigh the benefits. Sometimes it is easier to encode a behavior by hand. Sometimes a mathematical model is inadequately interpretable for a given situation. A machine learning technique may be too data intensive, and limit the ability of the system to actually train.

Despite the challenges to machine learning, in this lab we are tasked with making our robot learn. We have freedom in choosing what the robot will learn, how it will learn it, and how we will measure it to assess performance.

II. METHODS

In this lab our Bioloid robot has a simple physical setup: two wheels mounted on the sides of a rectangular prism body with a Bluetooth controller mounted on top. Its task will be to learn how to move to a goal position in a smooth arc. As it moves, its heading will incrementally change; it will not simply adjust its heading to point toward the goal position and go straight. Each arc to a given position has a pair of wheel speeds that stays constant through the course of the movement; our robot must predict the correct wheel speeds.

The machine learning technique we will use is locally weighted regression. The function that maps positions to wheel speeds is not linear, so we need a technique like locally weighted regression (LWR) that is non-parametric (does not assume a certain structure of the model parameters). LWR involves calculating a prediction for an input point based on training data points nearby. Local regression is "sometimes referred to as a memory-based procedure, because like nearest-neighbors, we need all the training data each time we wish to compute a prediction."[6]

Each time LWR makes a prediction, it assigns every data point in the training data a weight based on the input/target point; data closer to the target point gets a higher weight. In LWR one must choose how to calculate the weights, and there are a number of options. Our implementation uses a method called kernel smoothing, which allows us to "smooth out our point of interest using nearby data points."[7] In kernel smoothing, the kernel is a function whose value decreases as the distance between the target point and a given training data point increases. Our code uses a Gaussian kernel, defined by the following equation[8]:

$$D = ae^{-\frac{\|X - X_0\|}{2c}}$$

Figure 1: Gaussian kernel equation

---

[4] Matarić, p. 261

[5] Matarić, p. 260

[6] James et al., p. 281

[7] Vilkeliskis

[8] Vilkeliskis

Our code calculates this weight for each training data point and places it in a weights matrix, W. Then, our code finds the parameters for a local linear regression by calculating the following[9]:

$$\beta = (X'WX)^{-1}X'WY$$

Figure 2: equation for local regression

In this equation, X is the inputs matrix and Y is the outputs matrix. This gives us a regression that is local to the data and can be used to make a prediction.

The other choice our model has is the value of the "span," a concept in LWR that "controls the flexibility of the non-linear fit." To elaborate, "the smaller the value of [the span], the more local and wiggly will be our fit; alternatively, a very large value of [span] will lead to a global fit to the data using all of the training observations."[10] In our implementation, span is controlled by the constant *c* in the Gaussian kernel in Figure 1. *C* can lie in the range $0 < c \leq 1$. A lower value of c will give an overall regression curve that is wavier and more responsive to outlying training data points, while a value of 1 will give a simple least squares linear regression.

In the end, our model takes three inputs during each prediction: a training inputs matrix, with columns that represent the x and y coordinates of positions in the environment, a training outputs matrix, with two columns that represent the wheel speed pairs, and a tuple that represents the goal position. It outputs the predicted wheel speed pair, and then runs the robot at those wheel speeds for eight seconds, hopefully sending it to the goal location.

The first test we will run is the data collection process. To do this, we have a Python script that will generate random wheel speeds for the left and right wheels of the robot and then run the robot at those speeds for eight seconds. We will measure the position at which the robot ended and record the position and wheel speed pair as one row in the dataset. This test is important because without it, our robot will have no data from which to learn.

The second test will be a 5-fold cross validation of the model on our training data to determine the optimal *c* value in our Gaussian kernel. This will show us the optimal flexibility of the LWR for our particular dataset.

Our last test will be trials of the robot's ability to predict the wheel speeds that will send it in an arc to the goal position. We will choose a list of positions to test, and then input them one by one into the LWR algorithm, each time getting a prediction that moves the robot in an arc. At its final position, we will measure to see where the robot actually ended up and compare it with the goal. This will tell us how accurate the prediction was. It will show if the robot learned to be able to move to a goal position in one, smooth arc.

III. RESULTS

Test 1: Training data collection

First, we lined up the midpoint of the back of the robot's controller with an intersection in the square linoleum tile on the floor. We treated this point as the origin, (x, y) = (0, 0); the crack in the linoleum in line with the robot's heading was the y-axis and the crack perpendicular to the heading was the x-axis. To generate training data points, we wrote a Python script that randomly generated a wheel speed in the range 400 to 700 for each of the left and right wheels. It then set the wheel speeds of the robot to these speeds, waited for 8 seconds, and stopped the wheels. We measured with a yardstick the movement along the x-axis and y-axis, and recorded this as the position for the given pair of wheel speeds. We repeated this 80 times. See Figure 3 for a plot of all the points we collected.

---

[9] Vilkeliskis

[10] James et al., p. 281

Figure 3: all positions of training data; horizontal is x-axis, vertical is y-axis

Test 2: Cross-validation on training data to tune span parameter
We did this procedure for each of the four *c* values 0.2, 0.5, 0.8, and 1. Each time, we randomly split our dataset into two parts: one had four fifths of the original dataset and would serve as the training data, and the other had the remaining fifth and would serve as the testing data. For every data point in the testing data, we made a prediction of the wheel speeds, computed the distance of our prediction from the real data point, and averaged over all these distances to give an average error for each run. We ran the procedure five times, and then averaged that, to get an average error for each *c* value.

c value	0.2	0.5	0.8	1
run 1 average error	111.159	87.583	48.264	125.344
run 2 average error	161.629	71.741	108.957	75.473
run 3 average error	165.287	94.874	87.449	84.014
run 4 average error	80.587	57.816	98.198	52.11
run 5 average error	197.46	74.999	43.231	69.956
average error over all runs	143.224	77.4026	77.2198	81.3794

Table 1: average errors of predicted wheel speeds

Test 3: Prediction trials
We lined up the robot at the origin, as in test 1. Starting with the first position in a list of 14 arbitrary positions that we created, we input a position into the model and watched the robot move based on the predicted wheel speeds. We measured the final position as in test 1 and recorded it in a table of positions. In Figure 4, you can see pairs of the goal positions and the positions actually reached by the robot, connected by red line segments. The longer the line segment, the more inaccurate was the prediction of the wheel speeds.

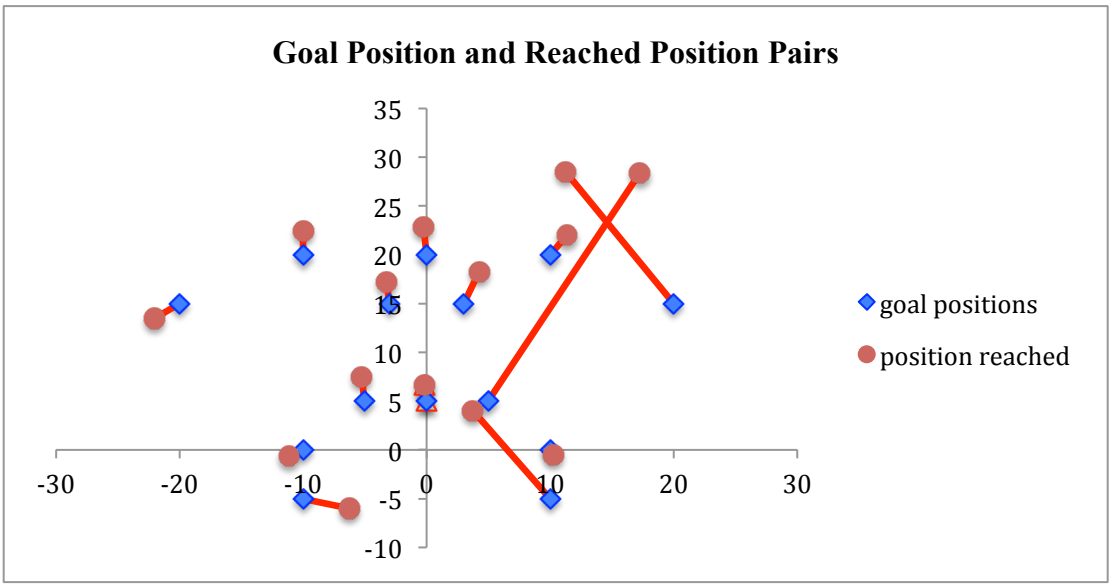

Figure 5

IV. DISCUSSION

Overall, we were pleased with the performance of our machine learning model in predicting wheel speeds for smooth arcs. Most of the trials achieved a final position quite close to the goal, as you can see in Figure 4. There were a few positions that performed very poorly; these can be seen in Figure 4 as the three long line segments. To us, there is no immediate explanation as to why these positions performed badly. However, we suspect that it could be due to the positions falling in a sparse part of our training dataset. With few nearby points, our LWR model would have drawn from further away points to make the prediction, causing the speeds to be off. Collecting more data would improve this.

The training data collection process for this task was inefficient. It was slow to input a position, sit there for 8 seconds, and then manually measure distances on the floor. This was a major limiting factor of the performance of our particular task, because although collecting more data would make the model more accurate, a researcher using our procedure would have to weigh the human-time costs of collecting the data. Luckily, our LWR model was not data-hungry enough that this inefficiency precluded accurate predictions; we still got some good ones. However, other more complicated techniques such as neural nets need more data than we would have been able to provide in this lab setting.

Because locally weighted regression requires all the training data every time it makes a prediction, it is a computationally intensive method. In our case, this was not a problem, because our dataset was so small, both in the parameter space and the number of data points. This would be a concern in more data intensive tasks, such as a machine learning model taking data from the internet, where data scales rapidly and may be high dimensional. In this situation, a researcher would need to do complexity analysis on the algorithms she or he uses to do machine learning, and LWR may not make the cut.

One of the central ideas in statistical/machine learning is bias-variance trade off. A simple least-squares regression is very biased; it assumes a linear structure to the data, which means that it generalizes well to data characterized by a linear function. Other learning methods that decrease bias by being flexible to nonlinear structure in the data, such as our LWR, increase variance. These methods may not generalize well, and react too much to changes in the training data. In our case, the c parameter in our Gaussian kernel directly affected the level of bias. Through our second test we found that the optimal c value was somewhere between 0.5 and 0.8. To us, this indicated that c values lower than 0.5 produced overfitting, where outlier values pulled the model in their direction, making predictions less accurate. It

would be interesting to see how the tuned value of the *c* parameter would change as our dataset grew; it may go down, as the model sees more data and more accurately assesses the underlying distribution.

V. CONCLUSION

In this lab assignment we built a locally weighted regression model that taught our robot to move to input goal positions in smooth arcs. Our robot performed fairly well in its predictions of the correct wheel speeds, but problems like the smallness of our dataset resulted in a few inaccurate predictions. Through the process, we learned the fundamentals of machine learning, and we saw how researchers might begin to program the intelligent robots of today that can learn all manner of tasks.

REFERENCES CITED

"Bioloid Premium Robot Kit." *Trossen Robotics*. N.p., n.d. Web. 03 Oct. 2016.

James, Gareth, Daniela Witten, Trevor Hastie, and Robert Tibshirani. "Moving Beyond Linearity." *An Introduction to Statistical Learning: With Applications in R*. New York: Springer, 2013. N. pag. Print.

Matarić, Maja J. *The Robotics Primer*. Cambridge, MA: MIT, 2007. Print.

Vilkeliskis, Tadas. "Machine Learning Notes—Locally Weighted Regression." *Vilkeliskis.com*. N.p., 8 Sept. 2013. Web. 04 Dec. 2016.