16-264 Humanoids HW2

Due Feb 23, 2022

Introduction

This assignment investigates learning the kinematics for a simple two-link robot arm. You will first derive the robot kinematics by hand, and then try learning them through a nearest neighbors method and a neural network method. This assignment leverages NumPy and PyTorch. Quick tutorials for these are available at https://numpy.org/doc/stable/user/quickstart.html and https://pytorch.org/tutorials/beginner/blitz/neural_networks_tutorial.html. As always, if you have any questions, feel free to come to my office hours or shoot me an email at mverghes@andrew.cmu.edu.

To get started, unzip the assignment (which I assume you have already done). Then open a terminal in this folder and enter "pip install -r requirements.txt" and hit enter.

Part 1: Kinematics

Here we will be deriving the forward kinematics of the two-link robot arm. The forward kinematics map from robot joint angles to the point in space of the end of the arm. As a reminder, here is what the two-link arm looks like:

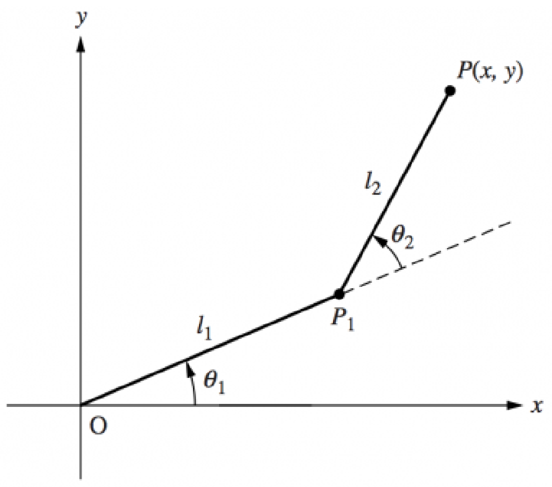

Figure 1: Diagram of a two-link robot arm

Fill out the method "Forward_Kinematics()" in the file "LearnKinematics.py". l_1 and l_2 are provided at the top of the file, and the functions "np.sin()" and "np.cos()" might be helpful. We provide the code for calculating inverse kinematics from forward kinematics. Visualize your forward kinematics with the provided code, is the arm able to reach the red target?

Part 2: Nearest Neighbors

Now that we have working forward kinematics, we can use it to generate data for our learning functions. In the real world, this might be done by moving the robot to various joint angles, and observing the position of its end. Fill out the function ”generate_data()”. This function should take N, the number of data points to generate, and create two arrays each $(N, 2)$ of corresponding joint angles and points.

Once we have a dataset we can now use it to try and learn the inverse kinematics of the robot. Fill out the function ”K_Nearest_Neighbors()”. It should take corresponding joint angle data and point data, and a query point, and return the estimated joint angles for that query point. The nearest neighbor method finds the closest point in the dataset to the query point, and gives its corresponding point as an answer. An example is shown below Use the provided code to visualize the results from this learned model, how well does it perform? Does more data help? What about using information from multiple nearest points (this is known as K Nearest Neighbors)?

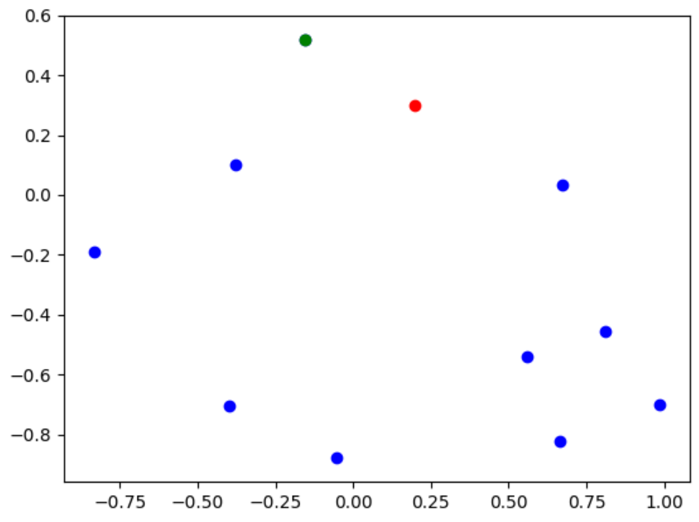

Figure 2: An example of nearest neighbors method. The dataset is in blue, the query point is in red, and the closest point in the dataset is in green.

Part 3: Neural Networks

Lastly, we will try training a Neural Net to learn kinematics. We have provided an example network with only two layers: a hidden layer with eight nodes, and an output layer with 2 nodes. This network can be found in the class ”ArmNet”. Experiment with more layers and other activation functions. Refer to the the PyTorch examples for adding more layers to the network. Note that you have to update both the init function, and the forward function. You can use the code to visualize the performance of your network. In addition to modifying the network itself, try changing the learning rate, batch size, and epochs. The learning rate controls how aggressively the network tries to optimize it's parameters. Higher rates will optimize faster, but may get stuck at a less optimal solution. The batch size controls how much of the data is used per update. For very large datasets, it is advisable to train on batches of the data rather than the entire dataset. Lastly the epochs control how long the network trains. After training the network will save its model. To pickup training where you left off, set "load=True". You can also experiment with how much data you give the network.

Part 4: Comparison

How did each of the models perform? Did they successfully learn the kinematics of the robot? Put together some slides on what you found. Include the parameters you experimented with and how they affected the results. What are the benefits and drawbacks of each of these models?

In practice, we sometimes try and use learning to improve on our existing models, rather than learn models from scratch. If our model is imperfect (which is likely with more complicated systems), we can use the mismatch between the expected results from the model and the real world results to learn a correction to our model. This correction is often referred to as a residual. If you want to, try messing up the kinematics slightly, and seeing if you can use either of our models to learn a residual. Is this easier than learning the model from scratch? If you do this, make sure you give the visualizer the correct arm lengths so it can plot the arm accurately.