Machine Learning Implementations and Analysis of Control for Inverted Pendulum Problem

Joshua Smith
ECE 6320
Utah State University
Logan, Utah 84322
joshua.smith4@aggiemail.usu.edu

***Abstract*— This paper serves as a tutorial and case study of several ML (machine learning) algorithms applied to the classic controls problem: the inverted pendulum. The derivation of the system model, accumulation of training data, training parameters of each ML algorithm, simulation results and instructions, and conclusions drawn from the experiments are detailed. Each ML algorithm is evaluated on its ability to balance the pendulum, robustness to system parameter changes, and performance with added noise. The simulation and Matlab code can be found at https://github.com/joshua-smith4/AIInvertedPendulumStudy.**

I. INTRODUCTION

Machine learning (ML) is a branch of artificial intelligence that focuses on algorithms and structures enabling machines to approximate hard-to-define functions by learning from experience. Experience is often found in the form of correctly labeled data representing the input and desired output of the function under scrutiny. Deep neural networks (DNN), decision trees/random forests, and support vector machines (SVM) are all machine learning techniques that can be applied to regression (function approximation) problems. These techniques have shown great performance in computer vision, speech recognition, general entity classification, and many other fields.

Applying machine learning to controls problems, such as the inverted pendulum, promises a few positive outcomes. One of these is the ability to reproduce the controls algorithm by treating it as a black box and reverse engineering it from data mapping the input to its output. Although other techniques, such as evolutionary algorithms, can be used to generate a control for a given model and defined cost function, this paper will deal strictly with the reverse engineering method to test the viability of neural networks, decision trees, and support vector machines to emulate a linear quadratic regulator (LQR) control.

II. MODELING THE SYSTEM

The inverted pendulum is a classic controls problem where there is a mass m connected to a cart with mass M by a massless rod. The objective is to balance the mass in the vertical upright position while being able to navigate the cart to a position x by applying a force u to the cart in the $\hat{i}$ direction. For simplicity, in this model the carts movement is restricted to the $\hat{i}$ direction.

Figure 1 shows the system variables and conventions assigned to the model of the inverted pendulum problem.

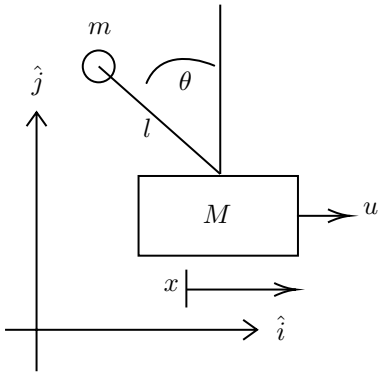

Fig. 1. Inverted pendulum model and naming convention definition.

Figure 2 is the free body diagram of forces applied to the cart. The value T is the magnitude of the tension force caused by the rod connecting the mass to the cart. N is the normal force that cancels with the force of gravity on the cart due to the restriction mentioned above of the carts movement in the $\hat{i}$ direction.

The free body diagram of the mass at the end of the rod is detailed in Figure 3.

Applying Newton's 2nd law to the free body diagrams we can derive Equations 1, 2, and 3. a_{px} and a_{py} are placeholder variables that represent the acceleration of the mass m in the x and y directions.

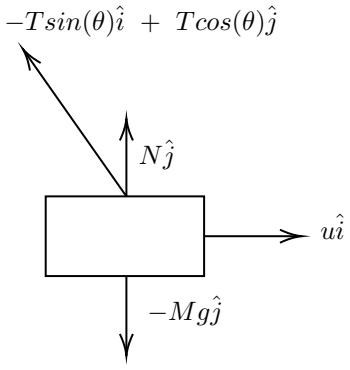

Fig. 2. Free body diagram of cart.

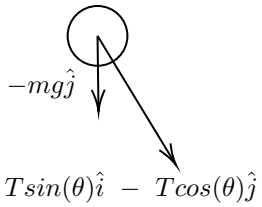

Fig. 3. Free body diagram of mass at end of massless rod.

$$u - Tsin(\theta) = M\ddot{x} \tag{1}$$

$$Tsin(\theta) = ma_{px} \tag{2}$$

$$-Tcos(\theta) - mg = ma_{py} \tag{3}$$

Equation 4 shows the relationship between the acceleration of the mass and the acceleration of the cart and transforms the equations that were in terms of a_{px} and a_{py} into a polar coordinate system in terms of $\hat{e_r}$ and $\hat{e_\theta}$.

$$a_p = a_c + a_{p/c} = \ddot{x} + l\ddot{\theta}\hat{e_\theta} - l\dot{\theta}^2\hat{e_r} \tag{4}$$

Equations 5 and 6 show the transformation necessary to get back to the coordinate frame of the original model: $\hat{i}$ and $\hat{j}$. Equation 7 is the substitution of equations 5 and 6 into Equation 4.

$$\hat{e_\theta} = -cos(\theta)\hat{i} - sin(\theta)\hat{j} \tag{5}$$

$$\hat{e_r} = -sin(\theta)\hat{i} + cos(\theta)\hat{j} \tag{6}$$

$$a_p = \ddot{x}\hat{i} + l\ddot{\theta}[-cos(\theta)\hat{i} - sin(\theta)\hat{j}] - l\dot{\theta}^2[-sin(\theta)\hat{i} + cos(\theta)\hat{j}] \tag{7}$$

Equations 8 and 9 separate Equation 7 into the $\hat{i}$ and $\hat{j}$ components that will be then substituted into Equations 2 and 3 as shown in Equations 10 and 11.

$$a_{px} = \ddot{x} - l\ddot{\theta}cos(\theta) + l\dot{\theta}^2sin(\theta) \tag{8}$$

$$a_{py} = -l\ddot{\theta}sin(\theta) - l\dot{\theta}^2cos(\theta) \tag{9}$$

$$Tsin(\theta) = m[\ddot{x} - l\ddot{\theta}cos(\theta) + l\dot{\theta}^2sin(\theta)] \tag{10}$$

$$-Tcos(\theta) - mg = m[-l\ddot{\theta}sin(\theta) - l\dot{\theta}^2cos(\theta)] \tag{11}$$

Equations 12, 13, and 14 detail the process used to eliminated the terms with the tension force T. Equation 10 is multiplied by $cos(\theta)$ and Equation 11 is multiplied by $sin(\theta)$ the the resulting equations are added together. As shown in Equation 14, the terms with T cancel out and the simplification in Equation 15 results in a very succinct equation in terms of g, l, θ, $\ddot{\theta}$, and $\ddot{x}$.

$$Tsin(\theta)cos(\theta) = m\ddot{x}cos(\theta) - ml\ddot{\theta}cos^2(\theta) + ml\dot{\theta}^2sin(\theta)cos(\theta) \tag{12}$$

$$-Tsin(\theta)cos(\theta) - mgsin(\theta) = -ml\ddot{\theta}sin^2(\theta) - ml\dot{\theta}^2sin(\theta)cos(\theta) \tag{13}$$

$$-mgsin(\theta) = m\ddot{x}cos(\theta) - ml\ddot{\theta}[cos^2(\theta) + sin^2(\theta)] \tag{14}$$

$$-gsin(\theta) = \ddot{x}cos(\theta) - l\ddot{\theta} \tag{15}$$

Equation 16 is the result of substituting Equation 10 into Equation 1 in order to eliminate the term with T.

$$u - ml\dot{\theta}^2sin(\theta) = (m+M)\ddot{x} - ml\ddot{\theta}cos(\theta) \tag{16}$$

In order to progress into linearizing this nonlinear system, it is important to arrange the two final equations (15 and 16) into the form $\dot{\vec{x}} = f(x,u)$. Equation 17 shows the definition of the states of the system and Equation 18 is its time derivative. Using these two definitions, we can now write the system equations we derived in the form $E\dot{\vec{x}} = G$ where E is defined by Equation 19 and G is defined by Equation 20.

$$\vec{x} = \begin{bmatrix} x \\ \theta \\ \dot{x} \\ \dot{\theta} \end{bmatrix} \tag{17}$$

$$\dot{\vec{x}} = \begin{bmatrix} \dot{x} \\ \dot{\theta} \\ \ddot{x} \\ \ddot{\theta} \end{bmatrix} \tag{18}$$

$$E = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & cos(\theta) & -l \\ 0 & 0 & m+M & -mlcos(\theta) \end{bmatrix} \tag{19}$$

$$G = \begin{bmatrix} \dot{x} \\ \dot{\theta} \\ -gsin(\theta) \\ u - ml\dot{\theta}^2sin(\theta) \end{bmatrix} \tag{20}$$

By inverting E and multiplying through with E^{-1}, the equation is put in the form $\dot{\vec{x}} = E^{-1}G = F = f(x,u)$ which is our desired form. F is defined by Equation 22.

$$\dot{\vec{x}} = E^{-1}G = F \tag{21}$$

$$F = \begin{bmatrix} \dot{x} \\ \dot{\theta} \\ \frac{u+gmcos(\theta)sin(\theta)-lm\dot{\theta}^2sin(\theta)}{M+m-mcos^2(\theta)} \\ \frac{ucos(\theta)+gmsin(\theta)+Mgsin(\theta)-lm\dot{\theta}^2cos(\theta)sin(\theta)}{l[M+m-mcos^2(\theta)]} \end{bmatrix} \tag{22}$$

The expressions in F fully define the dynamics of our system and can be used as a model for our simulations.

III. LINEARIZATION

In order to use linear systems theory and the LQR (linear quadratic regulator) controller, the nonlinear equations derived above must be linearized around the desired equilibrium point. The equilibrium point used represents the pendulum in the vertical upright position with no input force u. Equations 23 and 24 define these equilibrium parameters x_{eq} and u_{eq}. x_{eq} contains the variable x because the pendulum can be at equilibrium at any x coordinate so long as θ, $\dot{x}$, and $\dot{\theta}$ are all zero.

$$x_{eq} = \begin{bmatrix} x \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{23}$$

$$u_{eq} = 0 \tag{24}$$

The process of linearizing the equations and creating matrices A and B is shown by Equations 25 and 27 [2]. Applying the linearization process results in the A and B matricies defined by Equations 26 and 28.

$$A = \begin{bmatrix} \frac{\partial F(x_{eq},u_{eq})}{\partial x} & \frac{\partial F(x_{eq},u_{eq})}{\partial \theta} & \frac{\partial F(x_{eq},u_{eq})}{\partial \dot{x}} & \frac{\partial F(x_{eq},u_{eq})}{\partial \dot{\theta}} \end{bmatrix} \tag{25}$$

$$A = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & \frac{gm}{M} & 0 & 0 \\ 0 & \frac{g(m+M)}{Ml} & 0 & 0 \end{bmatrix} \tag{26}$$

$$B = \begin{bmatrix} \frac{\partial F(x_{eq},u_{eq})}{\partial u} \end{bmatrix} \tag{27}$$

$$B = \begin{bmatrix} 0 \\ 0 \\ \frac{1}{M} \\ \frac{1}{Ml} \end{bmatrix} \tag{28}$$

$$y = \begin{bmatrix} x \\ \theta \end{bmatrix} = C\vec{x} \tag{29}$$

The C matrix defined by Equation 30 is simply a result of the measured states from the simulation implemented in Simulink. x and θ are measured hence the associated 1s in C.

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \tag{30}$$

With the system correctly modeled and linearized around the equilibrium point, it is now ready for the LQR controller and for training data collection.

IV. LQR SOLUTION

The LQR controller will be used as our baseline comparison and each machine learning technique will be compared to it to determine its viability. LQR minimizes a cost function J as shown in Equation 31 over two parameters Q and R that satisfy the following conditions: $Q \geq 0$ and $R > 0$. Listing 1 shows the Matlab implementation of LQR used in the simulation and Equations 32 and 33 show the selected Q and R matrices. The resulting control signal is $u = -Kx$.

$$J(u) = \int_0^\infty \vec{x}^T Q\vec{x} + \vec{u}^T R\vec{u} \; dt \tag{31}$$

Listing 1. Syntax of Matlab lqr function

```
K = lqr(A,B,Q,R);
```

$$Q = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{32}$$

$$R = 1 \tag{33}$$

V. GATHERING TRAINING DATA

Collecting clean training data is critical to any good machine learning implementation. Figure 4 shows the Simulink model of the system and the measurement points on either side of the controller block that were used to collect training data. After running the simulation using the LQR controller for a simulation time of 1000 seconds, a time series of approximately 10,000 discrete values was saved into the two files x.mat and u.mat. The x (input) training data is a matrix of shape 10000×4 and the u (output) data is a matrix of shape 10000×1. Each row of x corresponds with a row in u and is used as the training data input/output pair in the regression problem.

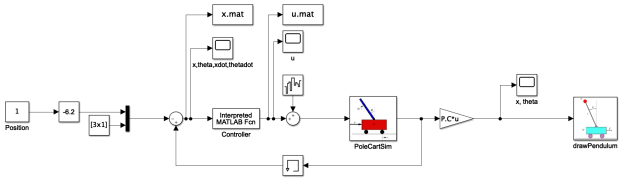

Fig. 4. Simulink model of system.

This process of collecting data was repeated twice for a total of 20,011 training data points. During collection, the equilibrium x value was shifted between the values of -5 and 5 in hopes that the machine learning algorithms could handle instantaneous shifts in this value of at least 10 units. Also during collection, the noise adding block shown after the controller in Figure 4 was zeroed out so that the ML techniques could be trained on clean data.

VI. NEURAL NETWORK ARCHITECTURE AND TRAINING

The network used in this experiment is relatively small compared to many modern day networks. Figure 5 shows the number of layers and neurons in each layer. The network is made up of purely fully-connected sigmoidal layers.

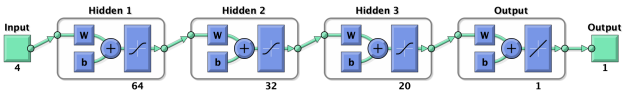

Fig. 5. Architecture of fully connected neural network.

Figure 6 shows the training statistics of the network. The Levenberg-Marquardt training algorithm was used and performance was measured with mean squared error (MSE) between the target output and the actual output. Only 10,005 of the 20,011 data points were used in the training of this network. As can be seen, the MSE became extremely small after 22 epochs (iterations over the training data) and the network was considered sufficiently trained. Also notable is the training time of 53 seconds which is extremely small compared to larger networks approximating more complex functions.

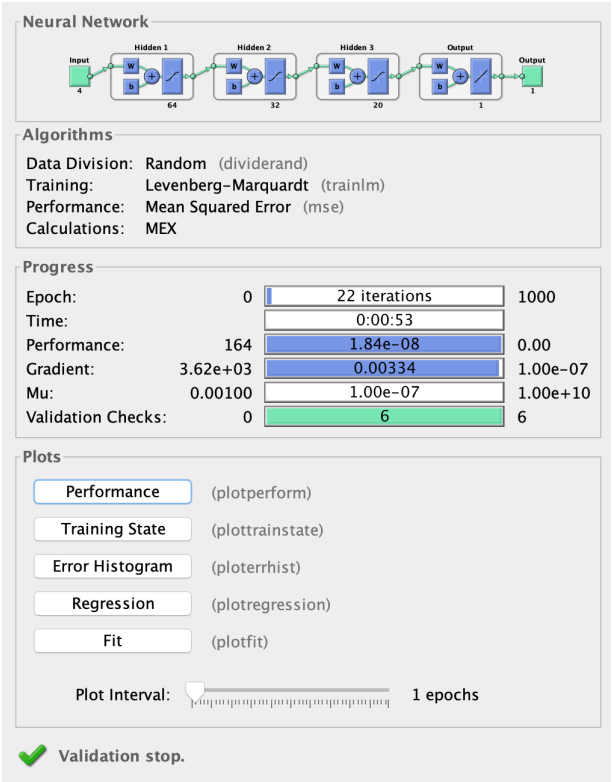

Fig. 6. Training statistics of neural network.

VII. RANDOM FOREST PARAMETERS

Decision trees are binary structures that split training data repeatedly into categories based on features in the data with the greatest information gain. Training data is formed into a tree structure which can be traversed with a new piece of data down to a leaf that is labeled with the appropriate output. In general they are extremely easy/fast to train but show less ability to generalize than neural networks or support vector machines.

Random forests, also called bags of trees, are collections of decision trees where the output is a result of each tree voting and majority rule. Random forests are generally more reliable and robust than individual trees. The number of trees in a random forests is a configurable parameters in their creation.

This experiment evaluates two different random forests called RFHalf and RFFull. RFHalf is trained with half the training data (10,005 data points) and RFFull is trained with all 20,011 data points. Each is a bag of 200 decision trees.

VIII. SUPPORT VECTOR MACHINE TRAINING

Support vector machines are less configurable than neural networks and random forests but tend to interpolate between data points well when presented with clean training data. The SVM used in this experiment was trained using 10,005 data points.

IX. SIMULATION INSTRUCTIONS

The project can be found in the following Github repository: https://github.com/joshua-smith4/AIInvertedPendulumStudy. The files shown in Table I, listed here with their description, can be found inside of the InvPend directory and are relevant to this experiment.

File	Description
controller.m	implementation of controller block
fc_net.m	function implementing the neural network controller
InvPendOnCart.slx	Simulink file used to run the simulation
loadData.m	function used to load training data
makeAndSaveRandomForest.m	wrapper for creating random forests
makeAndSaveSVM.m	wrapper for creating SVMs
param.m	file containing system parameters eg. selected controller, LQR solution
rand_forest_model.mat	file containing RFHalf
rand_forest_model_exp.mat	file containing RFFull
svm_model.mat	file containing SVM
trainFitNet.m	wrapper for creating neural networks
u_train[1,2].mat	output training data
x_train[1,2].mat	input training data

TABLE I

PROJECT FILES AND DESCRIPTIONS.

The code in Listing 2 should be run before running the Simulink simulation. After running this setup code and selecting the desired controller, open the InvPendOnCart.slx file in Simulink and press the play button. An animated view of the pendulum will appear. The selected controller can be changed at anytime by modifying the value of **P.controller**. By modifying the value in the band limited white noise block, the controller can be tested for robustness with different levels of noise added to its output signal. Also,

by modifying the value in the slider gain block, different equilibrium x values can be tested while the simulation is running to test the controller on its ability to handle different equilibrium points.

Listing 2. Workspace setup code.

```
param % loads parameter struct P
      % SVM, RFHalf, and RFFull
P.controller = 0 % select LQR
P.controller = 1 % select neural network
P.controller = 2 % select RFHalf
P.controller = 3 % select RFFull
P.controller = 4 % select SVM
```

X. RESULTS

Each controller was evaluated on three metrics. The first is a boolean representing ability to balance the pendulum with no noise or change to x_{eq}. The second is a measure of robustness to noise. The value in this field represents the max magnitude of the white noise signal added to the controller output for which the controller was still able to balance the pendulum for 100 seconds. The third is the max instantaneous change to x_{eq} the controller was able to handle with no noise added. Table II details the results of each of the controllers with respect to these three metrics.

Controller	Balances?	Noise Tolerance	Max x_{eq} Change
LQR	Yes	8.5	17
DNN	Yes	5.5	13.5
RFHalf	No	0	0
RFFull	Yes	0.2	10
SVM	Yes	6.5	17

TABLE II

RESULTS OF EACH CONTROLLER ON THREE METRICS.

LQR performed the best in noise tolerance and max x_{eq} change. This was to be expected as it would be somewhat improbable for the machine learning techniques to perform better than the controller they were trained on. The SVM modeled the LQR controller performance extremely well with just a slightly lower tolerance to noise. The DNN was next in line with a slight decrease in noise tolerance and max x_{eq} change. RFFull performed pretty well when no noise was added to the signal being able to handle a change of 10 to x_{eq}. It had nearly no noise tolerance. RFHalf was unable to balance the pendulum even without any noise.

XI. CONCLUSIONS

Given the performance of the SVM and neural network used, it can be seen that machine learning does have a place in controls. Decision trees seem to be unable to perform precise interpolation between training data points and therefore would be a weaker solution to fine or noisy systems. Increasing the training time of the neural network could also dramatically increase its performance but this is not guaranteed.

Each of these machine learning techniques had no access to the system details. The length of the pendulum or masses of the cart or ball were unknown. Even then, each technique was able to solve the problem. This is a powerful argument for the use of machine learning in controls.

The major downside of these ML algorithms is their inability to handle system parameter changes. Because they were trained on static parameters, they depend on static parameters and any significant change to the length of the rod or masses of the cart or ball causes them to fail. The LQR solution is able to adapt to an extremely wide range of these changes and therefore has a significant advantage.

The high dimensionality of these ML techniques also poses a problem because it is very difficult to prove the robustness of the solution. It has been shown that neural networks are susceptible to adversarial examples [1], small changes to the input that result in dramatic or unexpected changes to the output. This makes ML techniques a bad fit for safety-critical systems.

REFERENCES

[1] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. 2014.

[2] João P. Hespanha. *Linear Systems Theory*. Princeton Press, Princeton, New Jersey, February 2018. ISBN13: 9780691179575.