Machine Learning & Sensorimotor Control (Spring 2007)

Instructor: Dr. Sethu Vijayakumar

Homework 3

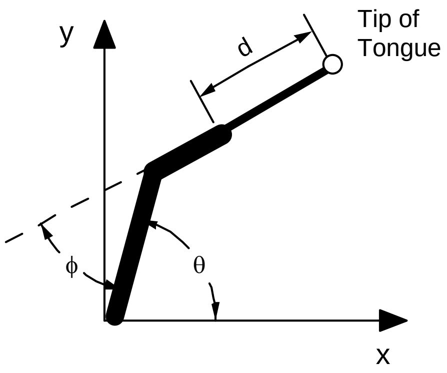
Due Date: April 10, 2006 (Tuesday)
Submission: JCMB 2107F

Instruction: Turn in short, precise and concise answers. Turn in all Matlab/C /C++ code (a synopsis of it will do if it is large sized) that is requested or that you used to calculate the requested values. MATLAB must be used for Question 1.

Grading: Maximum marks are 100. This will be scaled to account for 20% of your total course marks. No collaborations allowed. Homeworks are individual assignments and any attempts at plagiarism will be viewed very seriously.

Problem 1 (Distal Learning Problem): *65 points*

This assignment is about learning forward and inverse models with regression networks. In Computational Neuroscience, people have been interested in modeling the behavior of frogs for some time. Here, you are asked to help them to build a model about how a frog can learn to catch its prey.



A simplified model of the frog is shown on the left. The frog's body rotation is given by the angle θ, the head rotation by an angle ϕ, and the length of the tongue protrusion by the distance d. Thus, in order to catch prey successfully, the frog has to orient its head and body appropriately first, and then the tongue has to lash out such that the prey is caught.

First we want to learn a forward model of the frog's prey-catching behavior. The file X.data contains 200 random input data points (θ, ϕ, d), and the file T.data contains the corresponding 200 output data (x, y). Note that the output data are noisy. You are asked to find an appropriate feedforward neural network to model this data. You find a "skeleton" MATLAB script in nn_skeleton.m on the web page which you can use and modify for the following problems.

a) The MATLAB function to initialize a feedforward network is “newff”. The “newff” function requires that you specify:
   - the minimum and maximum values for each input,
   - the number of layers in your network,
   - the activation function in each layer,
   - the number of units in each layer.

   Make a choice for each of these points. Train the network using backpropagation algorithms. MATLAB offers you the following choices in “newff”:
   - traingd - Train feed-forward network with backpropagation.
   - trainbfg - Train feed-forward network with quasi-Newton backpropagation.
   - trainrp - Train feed-forward network with reslient backpropagation.
   - trainlm - Train feed-forward network with Levenberg-Marquardt.

   Additionally, “newff” allows you to specify whether you want to train without momentum (“learngd”) or with momentum (“learngdm”).

   Modify nn_skeleton.m to train your selected network. Try at least two of the above algorithms and compare their learning results.
   Generate a plot showing the decrease of the *normalized* mean-squared error as a function of training epochs. ***[5+5 points]***

Now the frog has to learn how to catch prey. For this purpose you need to learn an inverse model of the frog’s prey-catching behavior. The inverse model takes as input the target coordinates of the prey, (x_d, y_d), and outputs an appropriate set of “motor commands” (θ, ϕ, d) to reach the target. The file P.data contains a set of (x_d, y_d) data that the frog should learn.

One possible way to learn the inverse model is the direct inverse modeling approach. It just reverses the inputs and outputs of the forward model, i.e., T.data becomes the inputs data set, and X.data the output data set.

b) Using the nn_skeleton.m script, train a neural network (with your favorite training algorithm) to learn the direct inverse model by using the X.data and T.data data sets as training data. Generate a plot of the *normalized* mean-squared error as a function of training epochs. ***[10 points]***

In order to assess the quality of the frog’s inverse model, use the function frog.m from the web page to assess how well the frog is able to reach for a target (use “help frog” in MATLAB after frog.m is downloaded into your current working directory). Use P.data as input to your direct inverse network to generate motor commands for these targets. “frog.m” allows to convert these commands to the coordinates in x,y-space at which the frogs will actually snap.

c) Plot the P.data data superimposed with the actual x,y coordinates the frog snaps at. What is the average absolute error of the frog's prey catching behavior? ***[10 points]***

d) Assuming that the average fly is approximately 0.005-0.01 meters in diameter, do you think the frog will have a chance to survive? ***[5 points]***

e) Provide reasons (keywords suffice) why the direct inverse model network may be an inadequate approach to learning the fog's prey catching behavior. ***[5 points]***

f) Briefly sketch an approach how the frog could learn to perform better. Give reasons (keywords suffice) why your approach will perform better. ***[10 points]***

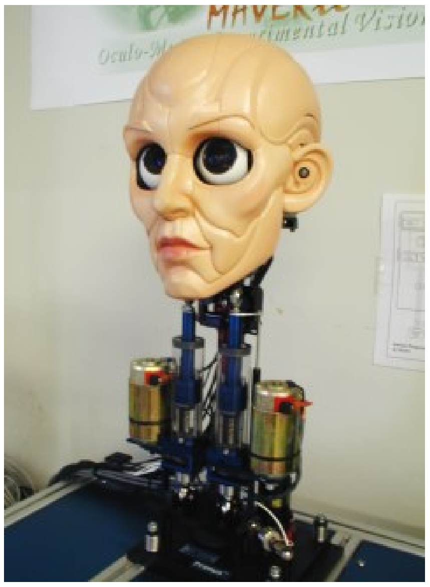
g) Use your alternative method from point (f) and implement this method in MATLAB. Provide a print-out of your MATLAB program, the learning curve of the system displaying how the error in x,y-space decreases over the learning epochs, and the average error of your method after learning has converged. Also, plot the P.data data superimposed with the actual x,y coordinates the frog snaps at. (Hint: The web-page contains a MATLAB function "frog_jac.m" which, given an input vector (θ, ϕ, d), returns the matrix of derivatives:

$$\begin{matrix} \frac{\partial x}{\partial \theta} & \frac{\partial x}{\partial \phi} & \frac{\partial x}{\partial d} \\ \frac{\partial y}{\partial \theta} & \frac{\partial y}{\partial \phi} & \frac{\partial y}{\partial d} \end{matrix}$$

As MATLAB does not provide a function how to calculate the Jacobian of a neural network, you can use this analytical function instead of the learned forward model to obtain the Jacobian. With the help of the Jacobian you can transform errors in distal space to proximal space. The errors in proximal space allows you to create new targets in proximal space that can be used to train the inverse model.) ***[15 points]***:

Problem 2 (Min. Jerk Trajectory Planning): *35 points*

This assignment is about planning kinematic trajectories based on certain optimization criterion. I am in charge of a new robotic vision head project. The manufacturer wants to finalize the specifications of the motor and hence, require me to tell them what the maximum velocity and maximum acceleration they need to design each joints for. I know that I will be implementing **minimum jerk trajectory** plans on the robot (in order to make the movement look smooth and nice!). I also know that the minimum jerk trajectories can be realized if I plan the movement of each joints as a fifth order spline (polynomial), i.e., $\theta(t) = b_0 + b_1 t + b_2 t^2 + b_3 t^3 + b_4 t^4 + b_5 t^5$, where t refers to the discreet time index and b_i are constants.



Let us assume the following constants:
Start position = θ_s, End position= θ_f, Start time = 0, End time = d, Amplitude of motion = a= (θ_f - θ_s).

(a) If we wanted to start a motion at zero velocity and zero acceleration and end also with zero velocity and zero acceleration, find the maximum velocity and the maximum acceleration at each joints in terms of the amplitude of motion=a and the duration of the movement=d; if we were to stick to min. jerk trajectories. ***[20 points]***

(b) Plot the joint position, velocity and acceleration profiles for the head rotation from zero to seventy-five degrees which is to be carried out in 0.4 sec. What would be the max. velocity and max. accelerations that this joint would have to be designed for. ***[15 points]***