DIVA Status Update [18Jun02]
satra

The DIVA model consists of the following components:

- A pseudoinverse controller consisting of a forward model and an inverse model
- A vocal tract model
- A hyperplane radial basis function network implementing the forward model (The 1998 paper had this network implementing the inverse model too, but since then we have moved to a simpler explicit pseudoinverse scheme. However, we will be providing an inverse model based on a supervised learning network very soon)
- Forward/Inverse network training
- A specification scheme (Formants, Formant ratios, muscle lengths, etc.)

DIVA currently has a modular structure based on a generic controller. This code does not have any GUI attached to it and lacks the translation capabilities that were in the older version (now with Mark). Specifically it lacks the components of translating from strings to target parameters and concatenated target sequencing. These can be easily put together in another module and for formants, it can be picked up from the old code.

The basic idea of this approach is that one has to write a pair of functions: a forward model and its corresponding inverse model. Fitting these functions into the controller drives the system towards its targets. In the current scenario, the inverse model is specifically a pseudoinverse model that maps derivatives to derivatives (or changes in one space to changes in another space).

Components of the model are described below.

<u>Maeda vocal tract synthesizer</u>
The latest version of the Maeda synthesizer is implemented as a Matlab object. The synthesizer class (d_opvt) takes as input vocal tract configuration (in terms of the Maeda vocal tract model), F0, Ag0 ang AgP (these parameters have default values for a steady vocal tract shape) and outputs various types of transformation such as the synthesized signal, LPC coefficients, validity of the vocal tract shape (in terms of producing an appropriate signal). In addition, the input parameters can be time varying such that an entire speech sequence is synthesized. The class also contains a display function, which plots the current vocal tract shape.

Code location: @d_opvt
Public functions: synth1, synth2, convert1, plot

<u>RBF Network</u>
Implements a hyperplane radial basis function network. The network implements three forms of network creation (3 ways of dealing with position of bases and spread) and two forms of network initialization (initialize output layer to pseudoinverse values/random values).

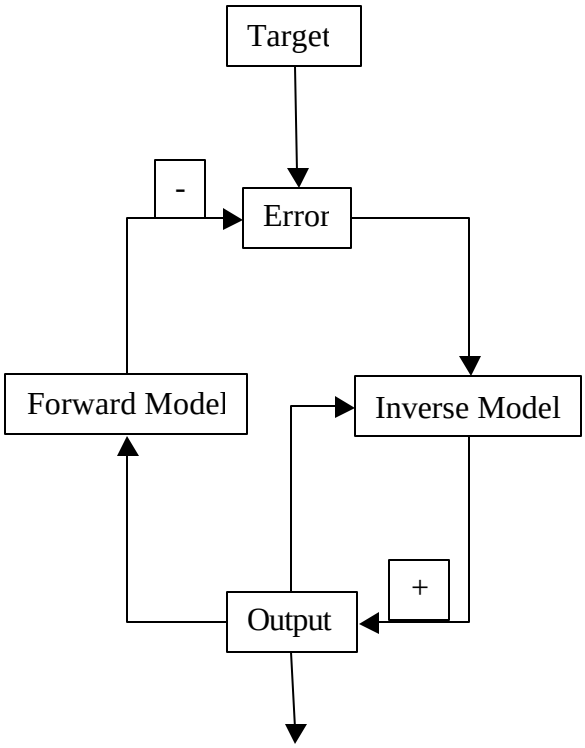
Code location: @ahrbf
Public functions: ahrbf (constructor), simrbf (simulate), trainrbf (train)

Controller

This implements a general forward-inverse controller that takes as input a forward function and inverse function along with other parameters such as matrix of current position, matrix of targets, control parameters for the forward and inverse functions. It then cycles in different ways to reach the target from different starting positions. For details see the code.

```
Example call:
r = controller01(cur_art,target_phoneme,@diva_forward01,@pseudoinv01,...
    tol,'fparams',fparams,'cparams',cparams,'iparams',iparams);
```

Code location: controller01.m



The figure above is the schematized controller used in the DIVA model. The model essentially performs a coordinate transform from the Target space to the Output space and is driven by the perceived error between the target desired and the current position in target space as determined by the forward model. The goal of the model is to reduce this thereby attaining a configuration in output space whose corresponding configuration in target space is within the tolerance limits of the target.

Forward Model

The forward model implemented is simulating the trained RBF network. The trained network learns the mapping from Maeda articulatory parameters to formant targets. The forward model is trained independently and then used for production. This can be reimplemented depending on what you want your specification to be. As long as it maintains the same input/output specifications, the general controller can be used to drive it.

Code location: diva_forward01.m, train_network.m

Inverse Model

The inverse model in this case is an explicit pseudoinverse controller. It determines the local direction of the gradient (Jacobian) at the current output position and then calculates the pseudoinverse based on that. A generic pseudoinverse controller has been implemented which is independent of model specification. It maps vectors of differences in the output space of the forward model to vectors of differences in the inputs space of the forward model, given the forward model. Thus the structure essentially allows any type of forward model to be used, as long as the output of the model is in the shape of a vector.

Similar to the forward model, once a network has been trained it can replace the explicit model (in progress).

Code location: pseudoinv01.m, train_invnetwork.m

Training details

The forward model is trained by generating valid samples in articulator space and the corresponding formants by using Maeda's vocal tract model. The validity of the samples is determined by certain properties of the generated formants.

- They have to be within normal ranges for F1, F2 and F3
- Two methods are used to determine the formants, and the outputs of each of these methods have to be within a certain tolerance level
- The vocal tract cannot be completely closed.

As mentioned before, only the forward model is trained at this point. We will provide routines for training the inverse model soon.

Code location: generate_data.m, generate_datarandn.m, train_network.m

TODO

There are a number of components still to be completed, which are specific to the type of target used.

- string to targets converter
- output parameter specification (F0, Ag0, AgP, the controller gives the change of Artic params) to be used with synth1/synth2 of the vocal tract model
- combining training and production into a simultaneous scenario, so train as it produces
- controller with more realistic temporal differences for different types of feedback
- speech recognition system/expert system to go hand in hand with simultaneous learning
- remove directory dependency from Maeda synthesizer/articulatory model

Summary

The current structure allows one to expand the model in terms of its functional components, without having to worry about control mechanisms too much. Constructing a working model is simply:

- To determine the kind of target space to use
- Create a module for translating strings to sequence of targets

- Write a forward function that takes as input Maeda articulatory space and outputs target space vectors
- Write a function that takes the calculated trajectory and synthesizes a proper waveform

Demo code

```
% Set the parameters
fparams = load('trainednet3000v2'); % load the HRBF network
cparams.limit = 4;                    % Limit maximum articulator variation to
4 units
iparams.DELTA = 0.5;
iparams.ALPHA = 0.1;
iparams.BETA  = 0.05;

cur_art = rand(7,1);    % Random starting configuration

target_phoneme = [300;870;2240]; % U as in boot
tol = [25;50;100];               % target range in Hz

% generate the movement
r = controller01(cur_art,target_phoneme,@diva_forward01,@pseudoinv01,...
    tol,'fparams',fparams,'cparams',cparams,'iparams',iparams);

art = [];
% get articulator trajectory
for i=1:size(r.act_pos,1),
    art(:,end+1) = cur_art(:,i);
    for j=1:size(r.act_pos,2),
        traj = r.act_pos{i,j};
        art(:,end+[1:(size(traj,2)-1)]) = traj(:,2:end);
    end;
end;
art = min(max(art,-3),3);

% Create vocal tract object
vt = d_opvt;

% plot the final position
figure('Doublebuffer','on');
for i=1:size(art,2),
    plot(vt,art(:,i),1);
    drawnow;
end;

% Synthesize the final signal
fs = 16000;
sound_vec = synth1(vt,art(:,end),fs);
soundview(sound_vec,fs);
```