
- Using the skeleton codes, fill the following functions in ‘Fill’ folder:
  - Initialize neural network: initializeNN.m
  - Feedforward neural network: nnFF.m
  - Back-propagate the delta errors: nnBP.m
  - Activation functions: htan.m, sigmoid.m, ReLU.m
  - Derivatives of activation functions: dHtan.m, dSigmoid.m, dReLU.m

- Discuss the performance of RNN for the following issues:
  i) Activation functions: sigmoid, hyperbolic tangent, and rectified linear unit (ReLU)
  ii) The number of nodes in context units
  iii) The number of nodes in the hidden layer and the number of hidden layers
2) Implement an RNN with Parametric Bias (RNNPB) and test its performance for the given three kinds of trajectories. In the report, discuss about the RNNPB for the following issues:
   i) The number of PB nodes
   ii) The sensitivity of RNNPB to the PB values
   iii) How to utilize the PB values

3) Design your own RNN for fast convergence and accuracy
   - Submit the report along with main.m and the programmed functions by zip file name: HW4_yourname.zip

   - Due date: May 31, 2016
   - Send to: yhyoo@rit.kaist.ac.kr
HW #4. Recurrent Neural Network

- **RNN procedure**
  - Initialize the neural network (*initializeNN.m*)
  - Unfold the RNN in time by stacking identical copies of the RNN
  - Feedforward the unfolded RNN (*nnFF.m*)
  - Back-propagate the delta errors and update the weights of the RNN (*nnBP.m*)
  - Repeat the forward calculation and backpropagation
HW #4. Recurrent Neural Network

- **Results**
  - In the skeleton codes, there are 3 kinds of trajectories. (seq1.txt, seq2.txt, seq3.txt)
  - If you want to generate other trajectories, use the `targetGenerator1_v1` program.