EECS 484 Computational Intelligence, Fall 2018
Problem Set 4: Recurrent Neural Nets and Back-Propagation Through Time

In the present assignment, you should construct and train a neural net that receives a
"pacemaker" input (a regular "beat", which is high once every four time steps) and
produces a target output beat. Example code is provided. You will need to complete the
functions: compute_F_o(), compute_F_u() and compute_F_wji(), consistent with
Werbos' derivation.

Note the following definitions in the starter code. Neuron 1 is assumed to be a bias term,
which always outputs unity. The u-value (net input) and all synapses leading into this
neuron for all time steps are irrelevant. Similarly, neuron 2 is assumed to be the external
stimulus (the sync beat). Thus, the u-values for this neuron as well as all synapses
leading into this neuron are also irrelevant for all time steps. Neuron 3 is assumed to be
the output neuron (without loss of generalization). The output of this neuron is defined
for time 1 to match the target output value at time 1. For all time steps greater than 1, this
output should try to match the target outputs at the respective time steps. This code is
specialized for consideration of a single output. For all nodes greater than index 3, these
nodes are "interneurons"—neither bias nor sensory nor output. You should select the
number of interneurons to use. Note that the initial conditions for the interneurons may
be set to random outputs, but these are presumably not optimal. In addition to learning
improved synaptic weights, one can also learn more useful initial conditions for the
interneurons.

Also, note that the starter code assumes that the time delay from input to output is
inserted as follows. All neuronal outputs (sigmas) are defined for time 1. These lead to
computations of net inputs (u-values) at time 2. (U values at time 1 are irrelevant). The
u-values at time 2 lead to both phi primes and sigmas at time 2 (using the logsig()
activation function). That is, the activation function is assumed to have no time delay;
there is only a time delay from sigmas at time t implying u-values at time t+1. It will be
important to compute both u-values and sigma values up through time T_final.

Once you have completed the necessary 3 functions for this program, test your result
using the function test_dEsqd_dwji(). This function computes dE/dwji using
perturbations of each weight individually and performing a recomputation of error for
each weight change, thus computing the derivatives dE/dwji numerically. These should
compare favorably (exact at least to 4 decimal places) with the analytic derivatives from
your ordered derivative computations. (If not, look for bugs in your code). Once you
have dE/dwji debugged, be sure to comment out future calls to test_dEsqd_dwji(), as this
function is slow.

Once your code is working, train recurrent nets on two example target sets: beats.dat and
beats2.dat. This data corresponds to a single input and a single output. The inputs
consist of a regular "heartbeat", and the outputs are a responsive pattern. For beats.dat,
the input is (1,0,0,0) in a repeating pattern and the desired output is (0,0,1,1), in a

repeating pattern. For beats2.dat, the input is the same, but the desired output repeats the pattern: (0,0,1,1,0,0,1,0).

Beats.dat can be trained using a single interneuron (total number of neurons=4, including bias, input, output and 1 hidden neuron). For Beats2.dat, you will need more hidden neurons. Report on how many are needed and plot out your results vs. the target outputs.

Note: in performing training, you may want to tune the value for the learning factor used in the gradient descent, or apply an adaptive learning factor. Report on your convergence rates (how many iterations were required) and on what learning factors you recommend.