GTNN_Headless_Sim (Calls: 1, Time: 529.136 s)

Generated 27-Jan-2025 13:04:49 using performance time.
Function in file C:\Users\zaida\Desktop\Thesis\GTNN\GTNN_clean\old tool\GTNN_Headless_Sim.m
Copy to new window for comparing multiple runs

Parents (calling functions)

No parent

Lines that take the most time

Line Number	Code	Calls	Total Time (s)	% Time	Time Plot

160	`Q = Q + 0.5*eta * M .* ((Psip - Psin)*(vp - vn)…`	1000	462.763	87.5%	
146	`Gn = vn + netI - Q*(vp - vn) + Psin;`	1000	33.538	6.3%	
145	`Gp = vp - netI + Q*(vp - vn) + Psip;`	1000	30.770	5.8%	
45	`Q = 0.5*randn(nNeuron,nNeuron);`	1	0.896	0.2%	
192	`colorbar; title('Final Connectivity (Q+I)');`	1	0.208	0.0%	
All other lines			0.961	0.2%	
Totals			529.136	100%	

Children (called functions)

Code Analyzer results

Coverage results

Total lines in function	203

Non-code lines (comments, blank lines)	75
Code lines (lines that can run)	128
Code lines that did run	93
Code lines that did not run	35
Coverage (did run/can run)	72.66 %

Function listing

Time	Calls	Line

```
                1  function GTNN_Headless_Sim()
                2  % GTNN_Headless_Sim - A “headless” version of the GTNN demo that
                3  % runs the simulation purely in code, with no GUI or figures.
                4
                5  % ---------------------- USER PARAMETERS ---------------------------
< 0.001      1  6  nNeuron       = 10000;        % Number of neurons
< 0.001      1  7  useGPU        = false;     % Set this to 'true' if you want to run on the GPU
< 0.001      1  8  plotMembrane  = true;    % If you only want to store results, not plot them
< 0.001      1  9  T             = 1000;     % Total simulation steps (in "ms" units)
< 0.001      1 10  nSpeed        = 1;        % Speed multiplier (like the slider in the GUI)
< 0.001      1 11  dt            = 0.001;    % Simulation timestep
< 0.001      1 12  tau           = 0.01;     % Time constant
< 0.001      1 13  eta           = 0.1;      % Learning rate
< 0.001      1 14  learnFlag     = true;     % Whether to enable learning or not
< 0.001      1 15  dataflag      = false;    % If true, will use external 'userdata.mat'
< 0.001      1 16  repeatdata    = 100;      % # timesteps each data vector is applied
               17
               18 % If not using external data, specify any DC/AC drive here:
< 0.001      1 19  I_input       = zeros(nNeuron,1);   % DC input
               20
               21 %Example stimulation on neuron 3:
< 0.001      1 22  I_input(3)    = 0.5;
               23
< 0.001      1 24  ac_amp        = zeros(nNeuron,1);   % AC amplitude
< 0.001      1 25  freq          = 5*ones(nNeuron,1);  % AC frequency
< 0.001      1 26  a             = ones(nNeuron,1);    % "alpha" parameter for each neuron
               27 % ---------------------------------------------------------------------
               28
               29 %  If you want to load user data for training:
< 0.001      1 30  if dataflag
               31      load userdata.mat;        % This file must be in the same folder
               32      [dataLen,dataDim] = size(userdata);
< 0.001      1 33  else
< 0.001      1 34      userdata  = [];
< 0.001      1 35      dataLen = 0;
< 0.001      1 36      dataDim = 0;
< 0.001      1 37  end
               38
               39 % Initialize synaptic weight matrix Q, connectivity mask M, identity I
< 0.001      1 40  Q = zeros(nNeuron,nNeuron);
  0.103      1 41  M = ones(nNeuron,nNeuron);
  0.008      1 42  I = eye(nNeuron);
               43
               44 % (Optional) pick any custom Q here.  E.g. uncomment:
  0.896      1 45   Q = 0.5*randn(nNeuron,nNeuron);
               46 % Q(logical(eye(size(Q)))) = 0;  % zero diagonal
               47
               48 % Make sure everything is on the GPU if desired:
< 0.001      1 49  if useGPU
```

```
                50      Q       = gpuArray(Q);
                51      M       = gpuArray(M);
                52      I       = gpuArray(I);
                53      I_input = gpuArray(I_input);
                54      a       = gpuArray(a);
                55      ac_amp  = gpuArray(ac_amp);
                56      freq    = gpuArray(freq);
< 0.001      1  57  end
                58
                59  % State variables
< 0.001      1  60  vp   = -0.5*ones(nNeuron,1,'like',Q);  % Positive membrane potential branch
< 0.001      1  61  vn   = -0.5*ones(nNeuron,1,'like',Q);  % Negative membrane potential branch
< 0.001      1  62  Psip = zeros(nNeuron,1,'like',Q);      % +spike flags
< 0.001      1  63  Psin = zeros(nNeuron,1,'like',Q);      % -spike flags
                64
                65  % Logging for energy/spikes.  We store the spiking energy in S_av, etc.
< 0.001      1  66  win        = 900;           % time window for smoothing
< 0.001      1  67  S_hist     = zeros(1,win,'like',Q);
< 0.001      1  68  S_av       = zeros(1,T,'like',Q);
< 0.001      1  69  spikeEnergy = 0;
                70
                71  % Extra parameters from the GUI version
< 0.001      1  72  Lambda = 5;   % threshold parameter
< 0.001      1  73  vmax   = 1;   % max membrane potential
< 0.001      1  74  vth    = 0;   % threshold for spiking
< 0.001      1  75  C      = 1;   % amplitude assigned to Psip / Psin upon spike
                76
                77  % Create variables for user-data iteration
< 0.001      1  78  currentIndex  = 1;
< 0.001      1  79  currentCount  = 0;
< 0.001      1  80  output        = zeros(dataLen,1);  % if you're going to store data usage
                81
                82  % Pre-allocate a buffer for logging membrane potentials (optional).
                83  %  If you only need final Q or final spiking rates, you can skip this.
< 0.001      1  84  ylog = zeros(nNeuron, T, 'like', Q);
                85
                86  % --------------- MAIN SIMULATION LOOP ---------------
                87  % We do T steps, each possibly with multiple internal updates (nSpeed).
< 0.001      1  88  fprintf('Starting simulation with %d neurons for %d steps...\n',nNeuron,T);
< 0.001      1  89  iter = 1;
                90
< 0.001      1  91  for t = 1:T
                92
                93      % Repeat "nSpeed" times each ms-step if desired
  0.003   1000  94      for subIter = 1:nSpeed
                95
                96          % Decide on input current (from either data or user-specified AC/DC)
< 0.001   1000  97          if dataflag
                98              % If using training data from 'userdata'
```

```
                          if currentCount > repeatdata
                              if learnFlag
                                  currentIndex = randi(dataLen,1);   % pick random pattern
                              else
                                  currentIndex = currentIndex + 1;
                                  if currentIndex > dataLen
                                      currentIndex = 1;
                                  end
                              end
                              currentCount = 0;
                          end
                          currentCount = currentCount + 1;
                          if useGPU
                          netI = gpuArray(netI);
                          end

                          % "userdata" might have fewer dims than nNeuron, so pad
                          if dataDim >= nNeuron
                              netI = userdata(currentIndex,1:nNeuron).';
                          else
                              netI = [userdata(currentIndex,:), ...
                                      zeros(1,nNeuron - dataDim)].';
                          end
                          % Optionally add a constant offset:
                          netI(end) = 0.1;  % from the old GUI example
< 0.001    1000       else
                          % If not using external data, just AC+DC
  0.117    1000           netI = I_input + ac_amp .* sin(2*pi*freq*iter/1000);
< 0.001    1000       end

                      % Convert to GPU if we are using GPU mode:


                      % -- Core spiking logic from the original code --

                      % 1) Check for spiking
  0.012    1000       spikedP = (vp > vth);
  0.010    1000       spikedN = (vn > vth);
  0.004    1000       Psip(spikedP) = C;
  0.002    1000       Psin(spikedN) = C;
  0.001    1000       vp(spikedP)   = vth;
  0.001    1000       vn(spikedN)   = vth;

                      % 2) Optionally record the new membrane potential for debugging
  0.031    1000       ylog(:,t) = vp;   % or something else (like y in GUI code)
                      % 3) Calculate the gradient
 30.770    1000       Gp = vp - netI + Q*(vp - vn) + Psip;
 33.538    1000       Gn = vn + netI - Q*(vp - vn) + Psin;
                      % 4) Update vp, vn
```

```
0.079   1000  148          vp = a.*vp + (dt/tau)*(((vp.^2 - vmax^2).*Gp)./(-vp.*Gp + Lambda*vmax));
0.032   1000  149          vn = a.*vn + (dt/tau)*(((vn.^2 - vmax^2).*Gn)./(-vn.*Gn + Lambda*vmax));
              150
              151          % 5) Compute spiking "energy"
0.025   1000  152          numSpikes = sum(spikedP) + sum(spikedN);
0.039   1000  153          S_hist = [S_hist(2:end), numSpikes];   % shift in sliding window
0.002   1000  154          spikeEnergy = sum(S_hist)/(2*win*nNeuron);
< 0.001 1000  155          S_av(t) = spikeEnergy;
              156
              157          % 6) Learning
0.002   1000  158          if learnFlag
              159              %  Weight update (Psip-Psin)*(vp-vn)'
462.763 1000  160              Q = Q + 0.5*eta * M .* ((Psip - Psin)*(vp - vn)');
< 0.001 1000  161          end
              162
              163          % 7) Reset spike flags
0.021   1000  164          Psip(:) = 0;
0.007   1000  165          Psin(:) = 0;
              166
              167          % Optional: store some result if dataflag==1 and learnFlag==0
< 0.001 1000  168          if dataflag && ~learnFlag
              169              output(currentIndex) = spikeEnergy;
< 0.001 1000  170          end
              171
< 0.001 1000  172          iter = iter + 1;
< 0.001 1000  173      end % of sub-iterations
< 0.001 1000  174  end % of T steps
              175
              176  % Pull back GPU arrays to CPU memory if needed
< 0.001    1  177  if useGPU
              178      Q        = gather(Q);
              179      ylog     = gather(ylog);
              180      S_av     = gather(S_av);
< 0.001    1  181  end
              182
              183  % Report final connectivity matrix, energy, etc.
< 0.001    1  184  fprintf('Simulation complete.\n');
0.003      1  185  fprintf('Final mean spiking energy = %g.\n', mean(S_av(end-50:end)));
              186
              187  % If you want a quick plot at the end, optionally do it here:
< 0.001    1  188  if plotMembrane
0.055      1  189      figure;
0.090      1  190      subplot(1,2,1);
0.140      1  191      imagesc(Q + eye(nNeuron));
0.208      1  192      colorbar; title('Final Connectivity (Q+I)');
              193
0.024      1  194      subplot(1,2,2);
0.009      1  195      plot(S_av,'LineWidth',2);
< 0.001    1  196      set(gca,'YScale','log');
```

```
0.009    1  197      xlabel('Time step'); ylabel('Spiking Energy');
0.010    1  198      grid on; title('Spiking Energy (log scale)');
< 0.001  1  199  end
           200
           201  % If needed, save results
           202  % save('GTNN_results.mat','Q','ylog','S_av','-v7.3');
0.115    1  203  end
```

Local functions in this file are not included in this listing.