GTNN_Headless_Sim (Calls: 1, Time: 50.296 s)

Generated 27-Jan-2025 12:01:04 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	`if learnFlag`	1000	37.182	73.9%	
144		1000	3.929	7.8%	
145	`% 3) Calculate the gradient`	1000	3.896	7.7%	
43		1	0.903	1.8%	
148		1000	0.545	1.1%	
All other lines			3.840	7.6%	
Totals			50.296	100%	

Children (called functions)

Code Analyzer results

Coverage results

Total lines in function	205

Non-code lines (comments, blank lines)	77
Code lines (lines that can run)	128
Code lines that did run	101
Code lines that did not run	70
Coverage (did run/can run)	78.91 %

Function listing

Time	Calls	Line

```
< 0.001        1    1  function GTNN_Headless_Sim()
               1    2  % GTNN_Headless_Sim - A “headless” version of the GTNN demo that
               1    3  % runs the simulation purely in code, with no GUI or figures.
               1    4
               1    5  % ---------------------- USER PARAMETERS ---------------------------
< 0.001        1    6  nNeuron       = 10000;        % Number of neurons
< 0.001        1    7  useGPU        = true;     % 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
               1   17
               1   18  % If not using external data, specify any DC/AC drive here:
< 0.001        1   19  I_input       = zeros(nNeuron,1);   % DC input
               1   20
               1   21  %Example stimulation on neuron 3:
< 0.001        1   22  I_input(3)    = 0.5;
               1   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
               1   27  % ------------------------------------------------------------------
               1   28
               1   29  %  If you want to load user data for training:
< 0.001        1   30  if dataflag
               1   31      load userdata.mat;        % This file must be in the same folder
               1   32      [dataLen,dataDim] = size(userdata);
< 0.001        1   33  else
< 0.001        1   34      userdata  = [];
< 0.001        1   35      dataLen = 0;
               1   36      dataDim = 0;
               1   37  end
< 0.001        1   38
  0.109        1   39  % Initialize synaptic weight matrix Q, connectivity mask M, identity I
  0.009        1   40  Q = zeros(nNeuron,nNeuron);
               1   41  M = ones(nNeuron,nNeuron);
               1   42  I = eye(nNeuron);
  0.903        1   43
               1   44  % (Optional) pick any custom Q here.  E.g. uncomment:
               1   45   Q = 0.5*randn(nNeuron,nNeuron);
               1   46  % Q(logical(eye(size(Q)))) = 0;  % zero diagonal
< 0.001        1   47
  0.351        1   48  % Make sure everything is on the GPU if desired:
  0.230        1   49  if useGPU
```

```
0.322      1    50      Q       = gpuArray(Q);
< 0.001    1    51      M       = gpuArray(M);
< 0.001    1    52      I       = gpuArray(I);
< 0.001    1    53      I_input = gpuArray(I_input);
< 0.001    1    54      a       = gpuArray(a);
< 0.001    1    55      ac_amp  = gpuArray(ac_amp);
                56      freq    = gpuArray(freq);
                57  end
< 0.001    1    58
< 0.001    1    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
                62  Psip = zeros(nNeuron,1,'like',Q);      % +spike flags
                63  Psin = zeros(nNeuron,1,'like',Q);      % -spike flags
< 0.001    1    64
< 0.001    1    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);
                68  S_av       = zeros(1,T,'like',Q);
                69  spikeEnergy = 0;
< 0.001    1    70
< 0.001    1    71  % Extra parameters from the GUI version
< 0.001    1    72  Lambda = 5;   % threshold parameter
< 0.001    1    73  vmax   = 1;   % max membrane potential
                74  vth    = 0;   % threshold for spiking
                75  C      = 1;   % amplitude assigned to Psip / Psin upon spike
< 0.001    1    76
< 0.001    1    77  % Create variables for user-data iteration
< 0.001    1    78  currentIndex  = 1;
                79  currentCount  = 0;
                80  output        = zeros(dataLen,1);  % if you're going to store data usage
                81
  0.005    1    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.
                84  ylog = zeros(nNeuron, T, 'like', Q);
                85
< 0.001    1    86  % --------------- MAIN SIMULATION LOOP ---------------
< 0.001    1    87  % We do T steps, each possibly with multiple internal updates (nSpeed).
                88  fprintf('Starting simulation with %d neurons for %d steps...\n',nNeuron,T);
< 0.001    1    89  iter = 1;
                90
                91  for t = 1:T
  0.002 1000    92
                93      % Repeat "nSpeed" times each ms-step if desired
                94      for subIter = 1:nSpeed
< 0.001 1000    95
                96          % Decide on input current (from either data or user-specified AC/DC)
                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
< 0.001     1000    % Optionally add a constant offset:
                    netI(end) = 0.1;  % from the old GUI example
0.543       1000    else
< 0.001     1000        % If not using external data, just AC+DC
                        netI = I_input + ac_amp .* sin(2*pi*freq*iter/1000);
                    end

                    % Convert to GPU if we are using GPU mode:


                    % -- Core spiking logic from the original code --
0.077       1000
0.069       1000    % 1) Check for spiking
0.083       1000    spikedP = (vp > vth);
0.060       1000    spikedN = (vn > vth);
0.054       1000    Psip(spikedP) = C;
0.053       1000    Psin(spikedN) = C;
                    vp(spikedP)   = vth;
                    vn(spikedN)   = vth;
0.059       1000
                    % 2) Optionally record the new membrane potential for debugging
                    ylog(:,t) = vp;   % or something else (like y in GUI code)
3.929       1000
3.896       1000    % 3) Calculate the gradient
                    Gp = vp - netI + Q*(vp - vn) + Psip;
                    Gn = vn + netI - Q*(vp - vn) + Psin;
```

```
0.545   1000  148
0.401   1000  149          % 4) Update vp, vn
              150          vp = a.*vp + (dt/tau)*(((vp.^2 - vmax^2).*Gp)./(-vp.*Gp + Lambda*vmax));
              151          vn = a.*vn + (dt/tau)*(((vn.^2 - vmax^2).*Gn)./(-vn.*Gn + Lambda*vmax));
0.202   1000  152
0.064   1000  153          % 5) Compute spiking "energy"
0.107   1000  154          numSpikes = sum(spikedP) + sum(spikedN);
0.042   1000  155          S_hist = [S_hist(2:end), numSpikes];   % shift in sliding window
              156          spikeEnergy = sum(S_hist)/(2*win*nNeuron);
              157          S_av(t) = spikeEnergy;
< 0.001 1000  158
              159          % 6) Learning
37.182  1000  160          if learnFlag
< 0.001 1000  161              %  Weight update (Psip-Psin)*(vp-vn)'
              162              Q = Q + 0.5*eta * M .* ((Psip - Psin)*(vp - vn)');
              163          end
0.101   1000  164
0.071   1000  165          % 7) Reset spike flags
              166          Psip(:) = 0;
              167          Psin(:) = 0;
< 0.001 1000  168
              169          % Optional: store some result if dataflag==1 and learnFlag==0
< 0.001 1000  170          if dataflag && ~learnFlag
              171              output(currentIndex) = spikeEnergy;
< 0.001 1000  172          end
< 0.001 1000  173
< 0.001 1000  174          iter = iter + 1;
              175      end % of sub-iterations
              176  end % of T steps
< 0.001    1  177
0.176      1  178  % Pull back GPU arrays to CPU memory if needed
0.023      1  179  if useGPU
< 0.001    1  180      Q        = gather(Q);
< 0.001    1  181      ylog     = gather(ylog);
              182      S_av     = gather(S_av);
              183  end
< 0.001    1  184
0.003      1  185  % Report final connectivity matrix, energy, etc.
              186  fprintf('Simulation complete.\n');
              187  fprintf('Final mean spiking energy = %g.\n', mean(S_av(end-50:end)));
< 0.001    1  188
0.064      1  189  % If you want a quick plot at the end, optionally do it here:
0.093      1  190  if plotMembrane
0.142      1  191      figure;
0.202      1  192      subplot(1,2,1);
              193      imagesc(Q + eye(nNeuron));
0.025      1  194      colorbar; title('Final Connectivity (Q+I)');
0.009      1  195
< 0.001    1  196      subplot(1,2,2);
```

```
0.009    1  197      plot(S_av,'LineWidth',2);
0.010    1  198      set(gca,'YScale','log');
< 0.001  1  199      xlabel('Time step'); ylabel('Spiking Energy');
           200      grid on; title('Spiking Energy (log scale)');
           201  end
           202
0.065    1  203  % If needed, save results
           204  % save('GTNN_results.mat','Q','ylog','S_av','-v7.3');
           205  end
```

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