One-step error probability (weighted diagonal)

Erik Norlin

All equations are taken from the course book *Machine Learning With Neural Networks.*

```
pPatterns = [12,24,48,70,100,120];
nNeurons = 120;
nTrials = 10^5;
mRowWeightMatrix = zeros(1,nNeurons); % Row "m" from weight matrix "W" based on random
                                      % neuron "m" and all patterns for each "p"
tempRowWeightMatrix = zeros(1,nNeurons); % Row "m" from a temporary weight
                                         % matrix for each pattern
nSetOfPatterns = length(pPatterns);
oneStepErrorProb = zeros(1,nSetOfPatterns);

for iSetOfPatterns = 1:nSetOfPatterns
    nPatterns = pPatterns(iSetOfPatterns);
    patternsMatrix = zeros(nPatterns,nNeurons); % Matrix to store "p" number of patterns

    errorCounter = 0;
    for jTrial = 1:nTrials
        % Creating "p" number of random patterns
        for muPattern = 1:nPatterns
            for jNeuron = 1:nNeurons
                rand = randi(2); % Generating a random number between 1 and 2 as an
                                 % activation state of given neuron
                if rand == 2
                    rand = -1;
                end
                patternsMatrix(muPattern,jNeuron) = rand;
            end
        end
```

Hebb's rule to compute the weight matrix:

$$w_{ij} = \frac{1}{N}\sum_{\mu=1}^{p} x_i^{(\mu)} x_j^{(\mu)} \quad \text{for} \quad i \neq j, \quad w_{ii} = 0, \quad \text{and} \quad \theta_i = 0. \qquad (2.26)$$

```
        % Creating only the necessary row "m" from the weight matrix "W"
        % based on all generated patterns and a random chosen neuron "m"
        mRandomNeuron = randi(nNeurons);
        for muPattern = 1:nPatterns
            for jNeuron = 1:nNeurons
                tempRowWeightMatrix(jNeuron) = patternsMatrix(muPattern,mRandomNeuron)...
                * patternsMatrix(muPattern,jNeuron);
%                 if jNeuron == mRandomNeuron % wii = 0
%                     tempRowWeightMatrix(jNeuron) = 0;
%                 end
            end
            mRowWeightMatrix = mRowWeightMatrix + tempRowWeightMatrix;
        end
```

```
        mRowWeightMatrix = (1/nNeurons) * mRowWeightMatrix;
```

To compute activation state of chosen neuron:

$$s_i(t+1) = \begin{cases} g\left(\sum_j w_{mj} s_j(t) - \theta_m\right) & \text{for } i = m\,, \\ s_i(t) & \text{otherwise}. \end{cases} \qquad (1.9)$$

```
        % Choosing one random pattern to feed pattern to neuron "m" once, as one
        % asynchronous update (equation 1.9)
        randomPattern = randi(nPatterns);
        sInputPattern = patternsMatrix(randomPattern,:);
        sOutput = mRowWeightMatrix * sInputPattern';

        if sOutput < 0
            sOutput = -1;
        else % sOutput >= 0
            sOutput = 1;
        end

        % If the neuron is updated
        if sOutput ~= sInputPattern(mRandomNeuron)
            errorCounter = errorCounter + 1;
        end
    end
    oneStepErrorProb(iSetOfPatterns) = errorCounter / nTrials;
end

% Printing final result
disp("One-step error probability for each value of p:")
```

```
One-step error probability for each value of p:
```

```
for i = 1:nSetOfPatterns
    fprintf("p%d: %.4f\n", i, oneStepErrorProb(i));
end
```

```
p1: 0.0002
p2: 0.0032
p3: 0.0128
p4: 0.0182
p5: 0.0223
p6: 0.0231
```