Neural Networks

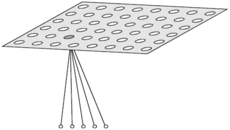
6. Self-Organizing Maps

Center for Cognitive Science
Department of Applied Informatics
Faculty of Mathematics, Physics and Informatics
Comenius University in Bratislava

Thursday 4^{th} April, 2024

Self-Organizing Map

- **Map:** grid of *rows* × *cols* neurons
  - other topologies possible, e.g. hexagonal grid



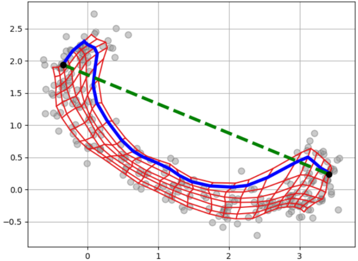
- **Inputs:** points in *n*-dimensional space:
  $\boldsymbol{x}_k \in \mathbb{R}^n$
- **Output:** position of winning neuron *in the grid*:
  $\boldsymbol{i} \in \{1, ..., rows\} \times \{1, ..., cols\}$
- **Weights:** positions of neurons in the *input space*:
  $\boldsymbol{W} \in \mathbb{R}^{rows \times cols \times n}$

Input space vs. Grid space

- **Input space:** $\mathcal{X} = \mathbb{R}^n$
  - input $\boldsymbol{x}_k \in \mathcal{X}$, neuron $\boldsymbol{w}_{\mathbf{i}} = \boldsymbol{w}_{[r,c,:]} \in \mathcal{X}$
  - distance: e.g. $\|\boldsymbol{x}_k - \boldsymbol{x}_j\|$ or $\|\boldsymbol{w}_{\mathbf{i}} - \boldsymbol{x}_k\|$
- **Grid space:** $\mathcal{G} = \{1, \ldots, rows\} \times \{1, \ldots, cols\}$
  - "index" of neuron = position in grid: $\boldsymbol{i} = (r_i, c_i) \in \mathcal{G}$
  - distance: $d(\boldsymbol{i}, \boldsymbol{j})$
- **Grid distance metrics:**
  - L_2-norm – *Euclidean* distance:
    $$d(\boldsymbol{i}, \boldsymbol{j}) = \sqrt{(r_i - r_j)^2 + (c_i - c_j)^2}$$
  - L_1-norm – *Manhattan* distance:
    $$d(\boldsymbol{i}, \boldsymbol{j}) = |r_i - r_j| + |c_i - c_j|$$
  - L_∞-norm – *axis-maximum* distance:
    $$d(\boldsymbol{i}, \boldsymbol{j}) = \max(|r_i - r_j|, |c_i - c_j|)$$

Input space vs. Grid space

Input space $\mathcal{X}$



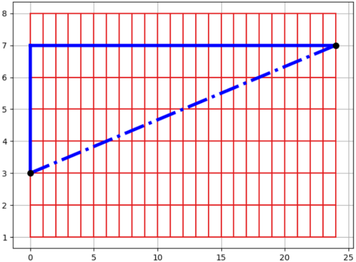
L_1 distance in $\mathcal{G}$

$d(\boldsymbol{i}, \boldsymbol{j}) = 28$

distance in $\mathcal{X}$

$\|\boldsymbol{w_i} - \boldsymbol{w_j}\| = 4.126$

Grid space $\mathcal{G}$



solid: L_1 distance in $\mathcal{G}$

$d(\boldsymbol{i}, \boldsymbol{j}) = 28$

dashed: L_2 distance in $\mathcal{G}$

$d(\boldsymbol{i}, \boldsymbol{j}) = 24.331$

Algorithm

- randomly initialize weights tensor $\boldsymbol{W}$
  - scale and shift weights to match inputs distribution
  - i.e. if inputs are in $[100, 135] \times [-10, 0]$, neurons positions should *not* be from $[0, 1] \times [0, 1]$.
- for each epoch $t \in \{0, 1, ..., t_{max} - 1\}$:
  for each input $\boldsymbol{x}$:
  - winner neuron $\boldsymbol{i}^*$
    $$\boldsymbol{i}^* = \arg\min_{\mathbf{i}} ||\boldsymbol{w}_{\mathbf{i}} - \boldsymbol{x}||$$
- adjust winner and its neighborhood:
  - "pull" neurons towards $\boldsymbol{x}$
  - h_t: neighborhood function
    $\Delta \boldsymbol{w}_{\mathbf{i}} = \alpha_t(\boldsymbol{x} - \boldsymbol{w}_{\mathbf{i}})h_t(\boldsymbol{i}, \boldsymbol{i}^*)$

Locality of Adjustments

- h_t: neighborhood function - inverted distance
  - $d(\boldsymbol{i}, \boldsymbol{i}^*) = 0 \implies h_t(\boldsymbol{i}, \boldsymbol{i}^*) = 1$
  - $d(\boldsymbol{i}, \boldsymbol{i}^*) = \infty \implies h_t(\boldsymbol{i}, \boldsymbol{i}^*) = 0$
- **Discrete neighborhood**:

$$h_t(\boldsymbol{i}, \boldsymbol{i}^*) = \begin{cases} 1 & \text{if } d(\boldsymbol{i}, \boldsymbol{i}^*) < \lambda_t \\ 0 & \text{otherwise} \end{cases}$$

  - only adjust close-enough neurons
- **Gaussian neighborhood** (continuous):
  - $h_t(\boldsymbol{i}, \boldsymbol{i}^*) = \exp\left(-\frac{d(\mathbf{i}, \mathbf{i}^*)^2}{\lambda_t^2}\right)$
  - adjust all neurons with a distance fall-off
  - winner gets full adjustment:
    $d(\boldsymbol{i}^*, \boldsymbol{i}^*) = 0 \implies h_t(\boldsymbol{i}^*, \boldsymbol{i}^*) = 1$

Training Parameters Schedule

- two training hyperparameters:
  - α_t: learning rate
  - λ_t: neighbourhood factor
- for first epoch, parameter value is $\alpha_0 = \alpha_s$ (start)
- for last epoch, parameter value is $\alpha_{t_{max}-1} = \alpha_f$ (finish)
- geometric schedule:
  - parameter value for epoch $t \in \{0, 1, ..., t_{max} - 1\}$ is:

$$\alpha_t = \alpha_s \cdot \left(\frac{\alpha_f}{\alpha_s}\right)^{\frac{t}{t_{max}-1}} \qquad \lambda_t = \lambda_s \cdot \left(\frac{\lambda_f}{\lambda_s}\right)^{\frac{t}{t_{max}-1}}$$

Task

- Train SOM to various datasets:
  - square/circle/ellipse - 2D
  - Iris dataset - 2D/3D/4D
- som.py TODO:
  - initialize weights and scale them to match inputs
  - find the winner neuron, return $\boldsymbol{i}^*$
  - α_t and λ_t schedule
  - discrete neighborhood
  - weight adjustment