Neural Networks

7. Self-Organizing Maps

Farkaš, Kuzma et al.

Center for Cognitive Science
Department of Applied Informatics
Faculty of Mathematics, Physics and Informatics UKBA

March 27th, 2018

Self-Organizing Maps

Self-Organizing Map

- **inputs:** n points in k-dimensional space:

$$\mathbf{x}_i \in \mathbb{R}^k; \quad \forall 1 \leq i \leq n$$

- **map:** $w \times h$ grid of neurons
  - other topologies possible, e.g. hex grid
- **output:** positions in this grid
- **parameters:** positions of neurons in the input space:

$$\mathbf{W} \in \mathbb{R}^{h \times w \times k}$$

Training: Parameter Schedule

- fixed number of epochs: t_{max}
  - i.e. time is $t \in \{1, 2, \ldots, t_{max}\}$
- for first epoch, parameter value is $\alpha_1 = \alpha_s$ (start)
- for last epoch, parameter value is $\alpha_{t_{max}} = \alpha_f$ (finish)

Training: Parameter Schedule

- fixed number of epochs: t_{max}
  - i.e. time is $t \in \{1, 2, \ldots, t_{max}\}$
- for first epoch, parameter value is $\alpha_1 = \alpha_s$ (start)
- for last epoch, parameter value is $\alpha_{t_{max}} = \alpha_f$ (finish)
- geometric schedule: parameter value for epoch t is:

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

Training: Algorithm

- for each epoch t, for each input x:
  - find winner neuron i^*:

    $$i^* = \arg\min_i \|\mathbf{w}_i - \mathbf{x}\|$$

  - **k-means**: adjust winner only:

    $$\Delta\mathbf{w}_{i^*} = \alpha_t \cdot (\mathbf{x} - \mathbf{w}_{i^*})$$

    - better E-M formulations of a k-means update exist (faster convergence, avoiding local minima)

  - **SOM**: adjust a neighborhood:

    $$\Delta\mathbf{w}_i = \alpha_t \cdot (\mathbf{x} - \mathbf{w}_i) \cdot q_t(i, i^*)$$

Training: Locality of Adjustments

- *in grid* distance between neurons:
  - current $i \mapsto (x_i, y_i)$ and winner $i^* \mapsto (x_{i^*}, y_{i^*})$

    $$d = d(i, i^*) = d\big((x_i, y_i), (x_{i^*}, y_{i^*})\big)$$
- limited neighbourhood (discrete):

  $$q_t(i, i^*) = \begin{cases} 1 & \text{if } d(i, i^*) < \lambda_t \\ 0 & \text{otherwise} \end{cases}$$
  - only adjust close-enough neurons
- gaussian neigbourhood (continuous):

  $$q_t(i, i^*) = \exp\left(-\frac{d(i,i^*)^2}{\lambda_t^2}\right)$$
  - adjust all neurons with a distance fall-off
  - winner gets full adjustment:

    $$d(i^*, i^*) = 0 \Rightarrow q_t(i^*, i^*) = 1$$

Training: Grid Distance Metrics

- L_2-norm – Euclidean distance:
  $$d = \left((x_i - x_{win})^2 + (y_i - y_{win})^2\right)^{\frac{1}{2}}$$
- L_1-norm – Manhattan distance:
  $$d = |x_i - x_{win}| + |y_i - y_{win}|$$
- L_∞-norm – axis-maximum distance:
  $$d = \max\left(|x_i - x_{win}|, |y_i - y_{win}|\right)$$
- or in general, L_p-norm
  $$d = \left(|x_i - x_{win}|^p + |y_i - y_{win}|^p\right)^{\frac{1}{p}}$$
  - only $1 \leq p$ is a norm, but $0 \leq p < 1$ gets used anyway