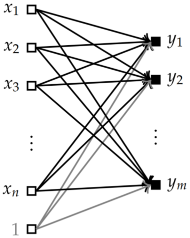
Neural Networks

3. Linear Models

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

Thursday 7th March, 2024

Single-layer perceptron



Input:	$\boldsymbol{x} = [x_1, x_2, ..., x_n]^T \in \mathbb{R}^n$
	$\boldsymbol{x'} = [x_1, x_2, ..., x_n, 1]^T \in \mathbb{R}^{n+1}$
Output:	$\boldsymbol{y} = [y_1, y_2, ..., y_m]^T \in \mathbb{R}^m$
Weights:	$\boldsymbol{W} \in \mathbb{R}^{m \times (n+1)}$
Computing output:	$\boldsymbol{y} = f(\boldsymbol{Wx'}) = f(\boldsymbol{net})$

One-hot encoding

- used for classification tasks
  $\boldsymbol{x}^{(p)} \rightarrow c^{(p)} \qquad c^{(p)} \in \{0, ..., m-1\}$

- to classify into m classes:
  - build a network with m output units
  - use target: $\boldsymbol{d}^{(p)} = \left[d_i \middle| 0 \leq i < m; d_i = \begin{cases} 1, & \text{if } i = c^{(p)} \\ 0, & \text{otherwise} \end{cases} \right]^T$
  - get class from network output $\hat{c}^{(p)} = argmax(\boldsymbol{y}^{(p)})$

- e.g. for $m = 6$ classes and $c^{(p)} = 3$:
  - $\boldsymbol{d}^{(p)} = [0, 0, 0, 1, 0, 0]$
  - $\boldsymbol{y}^{(p)} = [0.1, 0.1, 0, 0.2, 0.6, 0] \implies \hat{c}^{(p)} = 4$

Softmax

- Softmax function:
  $$y_i = f(\boldsymbol{net})_i = \frac{e^{net_i}}{\sum_k e^{net_k}}$$
- Computationally stable softmax:
  $$y_i = f(\boldsymbol{net})_i = \frac{e^{net_i - c}}{\sum_k e^{net_k - c}}; \quad c = max_k(net_k)$$
- Derivation of softmax:
  $$\frac{\partial y_i}{\partial net_j} = \begin{cases} y_i(1 - y_j), & \text{if } i = j \\ -y_i y_j, & \text{if } i \neq j \end{cases}$$
- Learning rule with Softmax + Cross-Entropy:
  $$w_{i,j}(t+1) = w_{i,j}(t) + \alpha(d_i - y_i)x'_j$$
  $$\boldsymbol{W}(t+1) = \boldsymbol{W}(t) + \alpha(\boldsymbol{d} - \boldsymbol{y})\boldsymbol{x'}^{\boldsymbol{T}}$$

Sequential vs. Batch

Sequential learning ("SGD")

- repeat N epochs:
  - for each input $\boldsymbol{x}^{(p)}$:
    - compute $\Delta\boldsymbol{W}$ from $\boldsymbol{x}^{(p)}$
    - $\boldsymbol{W} \leftarrow \boldsymbol{W} + \alpha\Delta\boldsymbol{W}$

Batch learning

- repeat N epochs:
  - compute cumulative $\Delta\boldsymbol{W}$ from all inputs $\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(P)}$ (using matrix operations to speed up computation)
  - $\boldsymbol{W} \leftarrow \boldsymbol{W} + \alpha\Delta\boldsymbol{W}$

Task

Complete missing parts of code in perceptron.py.

- Build single layer classificator with Softmax outputs
  - use sequential training
  - __init__, compute_output, train_seq
- Train classificator using batch training
  - train_batch
  - think about efficient implementation (using matrix operations and no cycles)