Inf2b Learning and Data

Lecture 14: Multi-layer neural networks (1)

Hiroshi Shimodaira
(Credit: Iain Murray and Steve Renals)

Centre for Speech Technology Research (CSTR)
School of Informatics
University of Edinburgh

Jan-Mar 2014

Today's Schedule

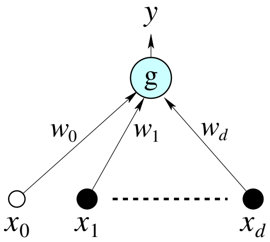
1. Single-layer network with a single output node (recap)
2. Single-layer network with multiple output nodes
3. Multi-layer neural network
4. Activation functions

Single-layer network with a single output node (recap)

- Activation function:

$$y = g(a) = g\left(\sum_{i=0}^{d} w_i x_i\right)$$

$$g(a) = \frac{1}{1 + \exp(-a)}$$



- Training set : $D = \{(\mathbf{x}^{(n)}, t^{(n)})\}_{n=1}^{N}$

  where $t^{(i)} \in \{0, 1\}$

- Error function:

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N} \left(y^{(n)} - t^{(n)} \right)^2$$

- Optimisation problem (training)

$$\min_{\mathbf{w}} E(\mathbf{w})$$

Training of single layer neural network

- Optimisation problem: $\min_{\mathbf{w}} E(\mathbf{w})$
- No analytic solution (no closed form)
- Employ an iterative method (requires initial values) e.g. Gradient descent (steepest descent), Newton's method, Conjugate gradient methods
- Gradient descent

  (scalar rep.)

$$w_i^{(\mathrm{new})} \leftarrow w_i - \eta \frac{\partial}{\partial w_i} E(\mathbf{w}), \qquad (\eta > 0)$$

  (vector rep.)

$$\mathbf{w}^{(\mathrm{new})} \leftarrow \mathbf{w} - \eta \nabla E(\mathbf{w}), \qquad (\eta > 0)$$

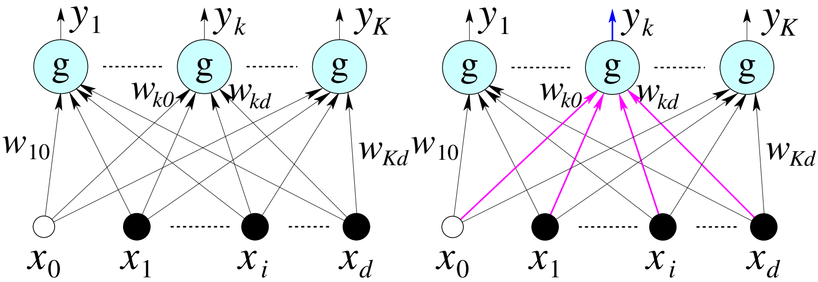
Training of the single-layer neural network

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\left(y^{(n)} - t^{(n)}\right)^2 = \frac{1}{2}\sum_{n=1}^{N}\left(g(a^{(n)}) - t^{(n)}\right)^2$$

where $a^{(n)} = \sum_{i=0}^{d} w_i x_i^{(n)}$. $\qquad \frac{\partial\, a^{(n)}}{\partial w_i} = x_i^{(n)}$

$$\begin{aligned}
\frac{\partial E(\mathbf{w})}{\partial w_i} &= \frac{\partial E(\mathbf{w})}{\partial y^{(n)}}\frac{\partial y^{(n)}}{\partial a^{(n)}}\frac{\partial a^{(n)}}{\partial w_i} \\
&= \sum_{n=1}^{N}(y^{(n)} - t^{(n)})\,\frac{\partial\, g(a^{(n)})}{\partial a^{(n)}}\,\frac{\partial\, a^{(n)}}{\partial w_i} \\
&= \sum_{n=1}^{N}(y^{(n)} - t^{(n)})\, g'(a^{(n)})\, x_i^{(n)}
\end{aligned}$$

Single-layer network with multiple output nodes



- K output nodes: $y_1, \ldots, y_K$.

$$y_k^{(n)} = g\left(\sum_{i=0}^{d} w_{ki}\, x_i^{(n)}\right) = g(a_k^{(n)})$$

$$a_k^{(n)} = \sum_{i=0}^{d} w_{ki}\, x_i^{(n)}$$

Single-layer network with multiple output nodes

- Training set : $D = \{(\mathbf{x}^{(1)}, \mathbf{t}^{(1)}), \ldots, (\mathbf{x}^{(N)}, \mathbf{t}^{(N)})\}$

  where $\mathbf{t}^{(n)} = (t_1^{(n)}, \ldots, t_K^{(n)})$ and $t_k^{(n)} \in \{0, 1\}$

- Error function:

$$E(w) = \frac{1}{2} \sum_{n=1}^{N} || \mathbf{y}^{(n)} - \mathbf{t}^{(n)} ||^2 = \frac{1}{2} \sum_{n=1}^{N} \sum_{k=1}^{K} (y_k^{(n)} - t_k^{(n)})^2$$

$$= \sum_{n=1}^{N} E^{(n)}, \quad \text{where} \quad E^{(n)} = \frac{1}{2} \sum_{k=1}^{K} (y_k^{(n)} - t_k^{(n)})^2$$

- Training by the gradient descent:

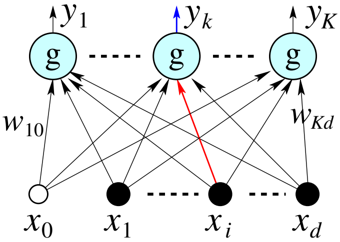
$$w_{ki} \leftarrow w_{ki} - \eta \frac{\partial E}{\partial w_{ki}}, \qquad (\eta > 0)$$

The derivatives of the error function (single-layer)

$$E^{(n)} = \frac{1}{2}\sum_{k=1}^{K}(y_k^{(n)} - t_k^{(n)})^2$$

$$y_k^{(n)} = g(a_k^{(n)})$$

$$a_k^{(n)} = \sum_{j=1}^{M} w_{kj} x_j^{(n)}$$



$$\frac{\partial E^{(n)}}{\partial w_{ki}} = \frac{\partial E^{(n)}}{\partial y_k^{(n)}}\frac{\partial y_k^{(n)}}{\partial a_k^{(n)}}\frac{\partial a_k^{(n)}}{\partial w_{ki}}$$

$$= (y_k^{(n)} - t_k^{(n)})\, g'(a_k^{(n)})\, x_i^{(n)}$$

Multi-layer neural networks

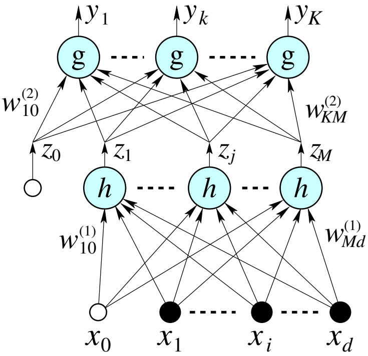
Multi-layer perceptron (MLP)

- Hidden-to-output weights:

$$w_{kj}^{[2]} \leftarrow w_{kj}^{[2]} - \eta \frac{\partial E}{\partial w_{kj}^{[2]}}$$

- Input-to-hidden weights:

$$w_{ji}^{[1]} \leftarrow w_{ji}^{[1]} - \eta \frac{\partial E}{\partial w_{ji}^{[1]}}$$



Training of MLP

1940s	Warren McCulloch and Walter Pitts : 'threshold logic' Donald Hebb : 'Hebbian learning'
1957	Frank Rosenblatt : 'Perceptron'
1969	Marvin Minsky and Seymour Papert : limitations of neural networks
1980	Kunihiro Fukushima: 'Neocognitoron'
1986	D. Rumelhart, G. Hinton, and R. Williams, "Learning representations by back-propagating errors" (1974, Paul Werbos)

The derivatives of the error function (two-layers)

$$E^{(n)} = \frac{1}{2}\sum_{k=1}^{K}(y_k^{(n)} - t_k^{(n)})^2$$

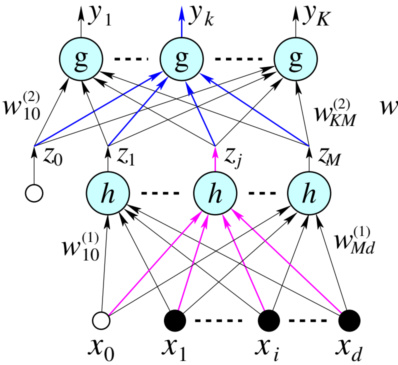
$$y_k^{(n)} = g(a_k^{(n)}), \quad a_k^{(n)} = \sum_{j=1}^{M} w_{kj}^{[2]} z_j^{(n)}$$

$$z_j^{(n)} = h(b_j^{(n)}), \quad b_j^{(n)} = \sum_{i=0}^{d} w_{ji}^{[1]} x_i^{(n)}$$

$$\frac{\partial E^{(n)}}{\partial w_{kj}^{[2]}} = \frac{\partial E^{(n)}}{\partial y_k^{(n)}} \frac{\partial y_k^{(n)}}{\partial a_k^{(n)}} \frac{\partial a_k^{(n)}}{\partial w_{kj}^{[2]}} = (y_k^{(n)} - t_k^{(n)})\, g'(a_k^{(n)})\, z_j^{(n)}$$

$$\frac{\partial E^{(n)}}{\partial w_{ji}^{[1]}} = \frac{\partial E^{(n)}}{\partial z_j^{(n)}} \frac{\partial z_j^{(n)}}{\partial b_j^{(n)}} \frac{\partial b_j^{(n)}}{\partial w_{ji}^{[1]}} = \left(\sum_{k=1}^{K}(y_k^{(n)} - t_k^{(n)}) \frac{\partial y_k^{(n)}}{\partial z_j^{(n)}}\right) h'(b_j^{(n)})\, x_i^{(n)}$$

$$= \left(\sum_{k=1}^{K}(y_k^{(n)} - t_k^{(n)}) g'(a_k^{(n)}) w_{kj}^{[2]}\right) h'(b_j^{(n)})\, x_i^{(n)}$$

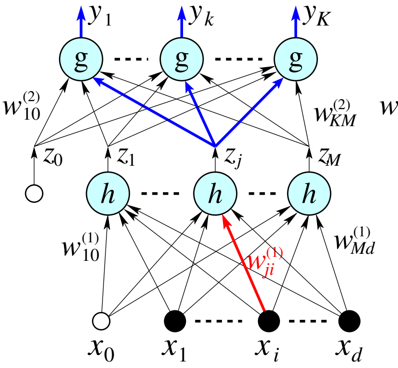


Error back propagation

$$\frac{\partial E^{(n)}}{\partial w_{kj}^{[2]}} = \frac{\partial E^{(n)}}{\partial y_k^{(n)}} \frac{\partial y_k^{(n)}}{\partial a_k^{(n)}} \frac{\partial a_k^{(n)}}{\partial w_{kj}^{[2]}}$$

$$= (y_k^{(n)} - t_k^{(n)})\, g'(a_k^{(n)})\, z_j^{(n)}$$

$$= \delta_k^{[2](n)}\, z_j^{(n)}, \qquad \delta_k^{[2](n)} = \frac{\partial E^{(n)}}{\partial a_k^{(n)}}$$

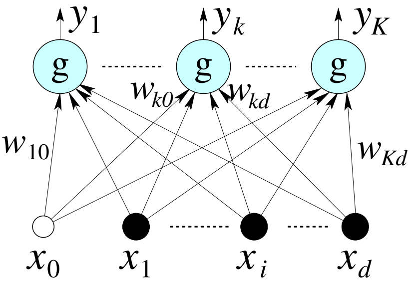


$$\frac{\partial E^{(n)}}{\partial w_{ji}^{[1]}} = \frac{\partial E^{(n)}}{\partial z_j^{(n)}} \frac{\partial z_j^{(n)}}{\partial b_j^{(n)}} \frac{\partial b_j^{(n)}}{\partial w_{ji}^{[1]}}$$

$$= \left(\sum_{k=1}^{K} (y_k^{(n)} - t_k^{(n)}) g'(a_k^{(n)}) w_{kj}^{[2]} \right) h'(b_j^{(n)})\, x_i^{(n)}$$

$$= \left(\sum_{k=1}^{K} \delta_k^{[2](n)} w_{kj}^{[2]} \right) h'(b_j^{(n)})\, x_i^{(n)}$$

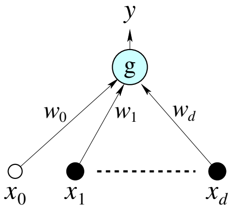
Notes on Activation functions



- Interpretation of output values
- Normalisation of the output values
- Other activation functions

Output of logistic sigmoid activation function

- Consider a single-layer network with a single output node logistic sigmoid activation function:

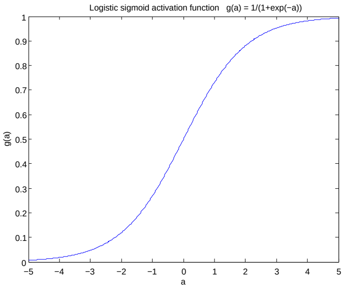


$$y = g(a) = \frac{1}{1+\exp(-a)} = g\left(\sum_{i=0}^{d} w_i x_i\right)$$
$$= \frac{1}{1+\exp\left(-\sum_{i=0}^{d} w_i x_i\right)}$$

- Consider a two class problem, with classes c_1 and c_2. The posterior probability of c_1:

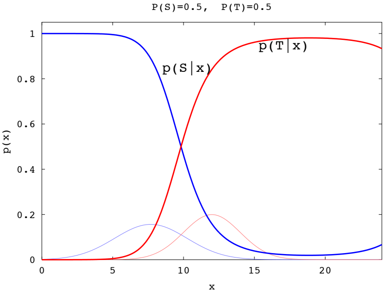
$$P(c_1|\mathbf{x}) = \frac{p(\mathbf{x}|c_1)\,P(c_1)}{p(\mathbf{x})} = \frac{p(\mathbf{x}|c_1)\,P(c_1)}{p(\mathbf{x}|c_1)\,P(c_1) + p(\mathbf{x}|c_2)\,P(c_2)}$$
$$= \frac{1}{1+\frac{p(\mathbf{x}|c_2)\,P(c_2)}{p(\mathbf{x}|c_1)\,P(c_1)}} = \frac{1}{1+\exp\left(-\ln\frac{p(\mathbf{x}|c_1)\,P(c_1)}{p(\mathbf{x}|c_2)\,P(c_2)}\right)}$$

Approximation of posterior probabilities



Logistic sigmoid function

$$g(a) = \frac{1}{1 + \exp(-a)}$$



Posterior probabilities of two classes with Gaussian distributions:

Normalisation of output nodes

- Original outputs:

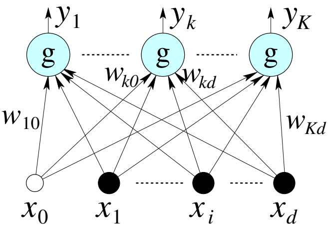
$$y_k = g(a_k), \quad a_k = \sum_{i=0}^{d} w_{ki} x_i$$

$$\left(\sum_{k=1}^{K} y_k\right) \neq 1$$

- Softmax activation function for $g()$:

$$y_k = \frac{\exp(a_k)}{\sum_{\ell=1}^{K} \exp(a_\ell)}$$

- Properties of the softmax
  (i) $0 \leq y_k \leq 1$
  (ii) $\sum_{k=1}^{K} y_k = 1$
  (iii) $y_k \approx P(c_k|\mathbf{x}) = \frac{p(\mathbf{x}|c_k)P(c_k)}{\sum_{\ell=1}^{K} p(\mathbf{x}|c_k)P(c_k)}$



Some questions on activation functions

- Is the logistic sigmoid function necessary for single-layer single-output-node network?
  - No, in terms of classification. (we can replace it with $g(a) = a$)
- What benefits are there in using the logistic sigmoid function?

Online gradient descent

$$E(w) = \frac{1}{2}\sum_{n=1}^{N} || \mathbf{y}^{(n)} - \mathbf{t}^{(n)} ||^2 \;=\; \frac{1}{2}\sum_{n=1}^{N}\sum_{k=1}^{K}(y_k^{(n)} - t_k^{(n)})^2$$

$$= \sum_{n=1}^{N} E^{(n)}, \quad \text{where} \;\; E^{(n)} = \frac{1}{2}\sum_{k=1}^{K}(y_k^{(n)} - t_k^{(n)})^2$$

- Batch gradient descent:

$$w_{ki} \;\leftarrow\; w_{ki} - \eta \frac{\partial E}{\partial w_{ki}}$$

- Incremental (online) gradient descent:
  Update weights for each $\mathbf{x}^{(n)}$

$$w_{ki} \;\leftarrow\; w_{ki} - \eta \frac{\partial E^{(n)}}{\partial w_{ki}}$$

- Stochastic gradient descent:
  Update weights for randomly chosen $\mathbf{x}$.

Summary

- Training of single-layer network
- Training of multi-layer network with 'error back propagation'
- Activation functions (e.g. softmax)