Inf2b - Learning

Lectures 12,13: Single layer Neural Networks (2,3)

Hiroshi Shimodaira
(Credit: Iain Murray and Steve Renals)

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

http://www.inf.ed.ac.uk/teaching/courses/inf2b/
https://piazza.com/ed.ac.uk/spring2020/infr08028
Office hours: Wednesdays at 14:00-15:00 in IF-3.04

Jan-Mar 2020

Today's Schedule

1. Perceptron (recap)
2. Problems with Perceptron
3. Extensions of Perceptron
4. Training of a single-layer neural network

Perceptron (recap)

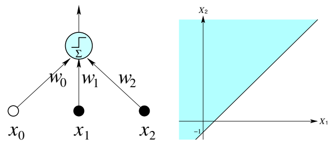
- Input-to-output function

$$a(\dot{\mathbf{x}}) = \boldsymbol{w}^T\boldsymbol{x} + w_0 = \dot{\mathbf{w}}^T\dot{\mathbf{x}}$$
$$\text{where } \dot{\mathbf{w}} = (w_0, \boldsymbol{w}^T)^T,\ \dot{\mathbf{x}} = (1, \boldsymbol{x}^T)^T$$

$x_0 = 1$

$$y(\dot{\mathbf{x}}) = g(\,a(\dot{\mathbf{x}})\,) = g(\,\dot{\mathbf{w}}^T\dot{\mathbf{x}}\,)$$
$$\text{where } g(a) = \begin{cases} 1, & \text{if } a \ge 0,\\ 0, & \text{if } a < 0 \end{cases}$$

$g(a)$: activation/transfer function



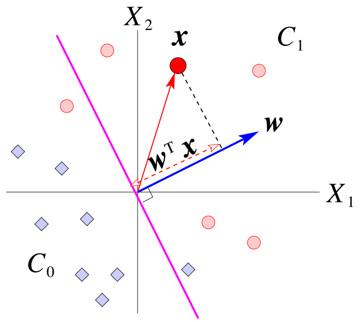
$x_2 \ge x_1 - 1$
$a(\boldsymbol{x}) = 1 - x_1 + x_2$
$= w_0 + w_1x_1 + w_2x_2$
$w_0 = 1,\ w_1 = -1,\ w_2 = 1$

Geometry of Perceptron's error correction

$$y(\mathbf{x}_i) = g(\boldsymbol{w}^T\mathbf{x}_i)$$
$$\boldsymbol{w}^{(\text{new})} \leftarrow \boldsymbol{w} + \eta\,(t_i - y(\mathbf{x}_i))\,\mathbf{x}_i \qquad (0 < \eta < 1)$$

$t_i - y(\mathbf{x}_i)$		$y(\mathbf{x}_i)$ 0	1
t_i	0	0	-1
	1	1	0

$\boldsymbol{w}^T\boldsymbol{x} = \|\boldsymbol{w}\|\|\boldsymbol{x}\|\cos\theta$

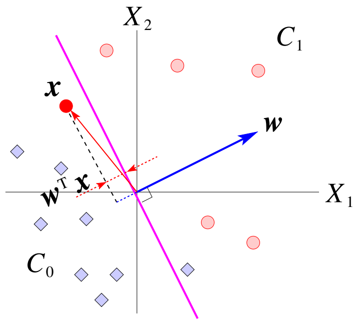


Geometry of Perceptron's error correction *(cont.)*

$$y(\mathbf{x}_i) = g(\boldsymbol{w}^T\mathbf{x}_i)$$
$$\boldsymbol{w}^{(\text{new})} \leftarrow \boldsymbol{w} + \eta\,(t_i - y(\mathbf{x}_i))\,\mathbf{x}_i \qquad (0 < \eta < 1)$$

$t_i - y(\mathbf{x}_i)$		$y(\mathbf{x}_i)$ 0	1
t_i	0	0	-1
	1	1	0

$\boldsymbol{w}^T\boldsymbol{x} = \|\boldsymbol{w}\|\|\boldsymbol{x}\|\cos\theta$

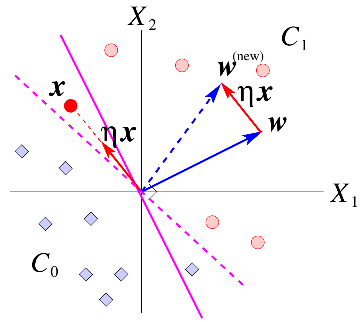


Geometry of Perceptron's error correction *(cont.)*

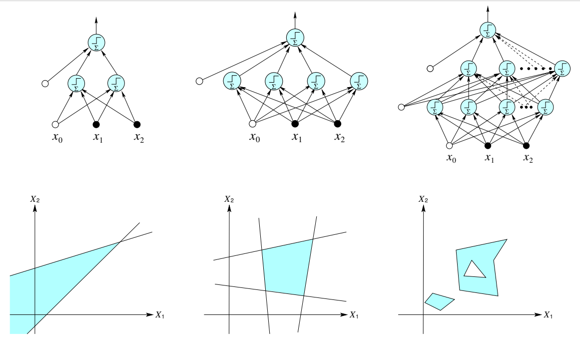
$$y(\mathbf{x}_i) = g(\boldsymbol{w}^T\mathbf{x}_i)$$
$$\boldsymbol{w}^{(\text{new})} \leftarrow \boldsymbol{w} + \eta\,(t_i - y(\mathbf{x}_i))\,\mathbf{x}_i \qquad (0 < \eta < 1)$$

$t_i - y(\mathbf{x}_i)$		$y(\mathbf{x}_i)$ 0	1
t_i	0	0	-1
	1	1	0

$\boldsymbol{w}^T\boldsymbol{x} = \|\boldsymbol{w}\|\|\boldsymbol{x}\|\cos\theta$



Perceptron structures and decision boundaries

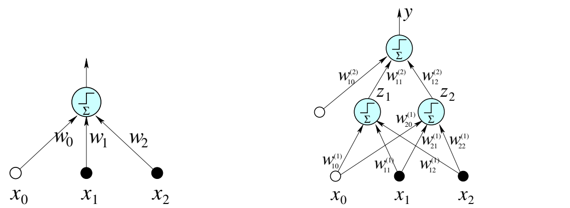


Question: Find the weights for each network

Limitations of Perceptron

- Single-layer perceptron is just a linear classifier (Marvin Minsky and Seymour Papert, 1969)
- Multi-layer perceptron can form complex decision boundaries (piecewise-linear), but it is hard to train
- Training does not stop if data are linearly non-separable
- Weights $\boldsymbol{w}$ are adjusted for misclassified data only (correctly classified data are not considered at all)

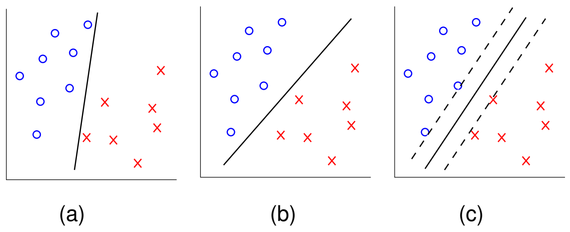
A limitation of Perceptron



$$y = g(\boldsymbol{w}^T\boldsymbol{x})$$

$$z_1 = g(\mathbf{w}_1^{(1)T}\boldsymbol{x}) = g(w_{11}^{(1)}x_1 + w_{12}^{(1)}x_2 + w_{10}^{(1)})$$
$$z_2 = g(\mathbf{w}_2^{(1)T}\boldsymbol{x}) = g(w_{21}^{(1)}x_1 + w_{22}^{(1)}x_2 + w_{20}^{(1)})$$
$$y = g(\mathbf{w}^{(2)T}\boldsymbol{z}) = g(w_{11}^{(2)}z_1 + w_{12}^{(2)}z_2 + w_{10}^{(2)})$$

Choices of decision boundaries



How can we resolve the problem of training?

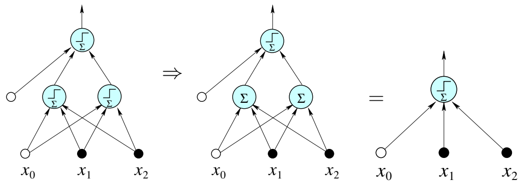
- Use the least squares error criterion for training

$$E_2(\boldsymbol{w}) = \sum_{n=1}^{N} (\, y_n - t_n \,)^2$$

- Replace $g(\,)$ with a differentiable function

  What about removing $g(\,)$ in the hidden layer?

$$z_i = g(\mathbf{w}_i^{(1)T}\boldsymbol{x}) \quad \Rightarrow \quad z_i = \mathbf{w}_i^{(1)T}\boldsymbol{x}$$



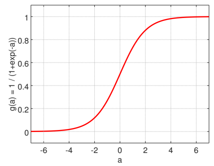
Question: Show networks with linear hidden nodes reduce to single-layer networks

How can we resolve the problem of training? *(cont.)*

- Replace $g(\,)$ with a differentiable non-linear function

  e.g., Logistic sigmoid function:

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



Mapping: $(-\infty, +\infty) \to (0, 1)$

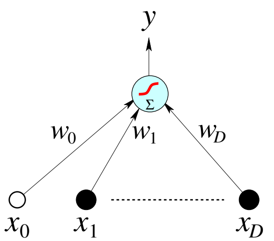
$$\frac{d}{da} g(a) = g'(a) = g(a)\,(1 - g(a))$$

Single Layer Neural Network

Assume a single-layer neural network with a single output node with a logistic sigmoid function:

$$y(\boldsymbol{x}) = g(\,\boldsymbol{w}^T\boldsymbol{x}\,) = g\left(\sum_{i=0}^{D} w_i x_i\right)$$

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



Single Layer Neural Network *(cont.)*

- Training set : $\mathcal{D} = \{(\mathbf{x}_1, t_1), \ldots, (\mathbf{x}_N, t_N)\}$

  where $t_i \in \{0, 1\}$

- Error function:

$$E(\boldsymbol{w}) = \frac{1}{2}\sum_{n=1}^{N} (\, y_n - t_n \,)^2$$

$$= \frac{1}{2}\sum_{n=1}^{N} \left(g(\boldsymbol{w}^T\mathbf{x}_n) - t_n\right)^2$$

$$= \frac{1}{2}\sum_{n=1}^{N} \left(g\left(\sum_{i=0}^{D} w_i x_{ni}\right) - t_n\right)^2$$

- Definition of the training problem as an optimisation problem

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

Training of single layer neural network

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

  (scalar rep.)

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

  (vector rep.)

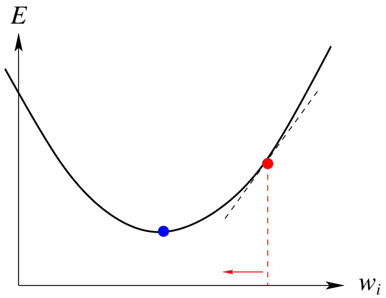
$$\boldsymbol{w}^{(\text{new})} \leftarrow \boldsymbol{w} - \eta \nabla_{\boldsymbol{w}} E(\boldsymbol{w}), \qquad (\eta > 0)$$

- Online/stochastic gradient descent (cf. Batch training)

  Update the weights one pattern at a time. (See Note 11)

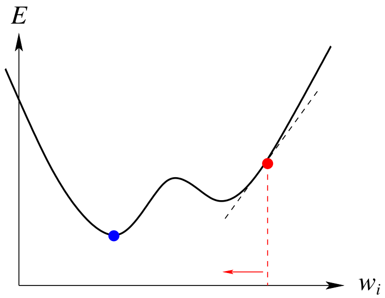
Gradient descent

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



Local minimum problem with the gradient descent

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



Training of the single-layer neural network

$$E(\boldsymbol{w}) = \frac{1}{2}\sum_{n=1}^{N} (\, y_n - t_n \,)^2 = \frac{1}{2}\sum_{n=1}^{N} \left(g\left(\sum_{i=0}^{D} w_i x_{ni}\right) - t_n \right)^2$$

where $y_n = g(a_n)$, $a_n = \sum_{i=0}^{D} w_i x_{ni}$, $\frac{\partial a_n}{\partial w_i} = x_{ni}$

$$\frac{\partial E(\boldsymbol{w})}{\partial w_i} = \frac{\partial E(\boldsymbol{w})}{\partial y_n}\frac{\partial y_n}{\partial a_n}\frac{\partial a_n}{\partial w_i}$$

$$= \sum_{n=1}^{N} (y_n - t_n) \frac{\partial g(a_n)}{\partial a_n}\frac{\partial a_n}{\partial w_i}$$

$$= \sum_{n=1}^{N} (y_n - t_n)\, g'(a_n)\, x_{ni}$$

$$= \sum_{n=1}^{N} (y_n - t_n)\, g(a_n)\,(1 - g(a_n))\, x_{ni}$$

Another training criterion – cross-entropy error

- Training problem with the mean squared error (MSE) criterion with the sigmoid function

$$E_{\mathrm{MSE}}(\boldsymbol{w}) = \frac{1}{2}\sum_{n=1}^{N}(y_n - t_n)^2\,, \quad y_n = g(a_n)$$

$$\frac{\partial E_{\mathrm{MSE}}(\boldsymbol{w})}{\partial w_i} = \sum_{n=1}^{N}(y_n - t_n)\,g'(a_n)\,x_{ni}\,, \quad g'(a) = g(a)(1-g(a))$$

For such a that $g(a) \approx 0$ or 1, $g'(a) \approx 0$.

- Cross-entropy error *(NE)*

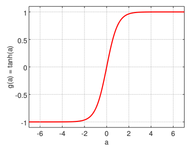
$$E_{\mathrm{H}}(\boldsymbol{w}) = -\frac{1}{N}\sum_{n=1}^{N}\{\,t_n \ln y_n + (1-t_n)\ln(1-y_n)\,\}$$

It can be shown that:

$$\frac{\partial E_{\mathrm{H}}(\boldsymbol{w})}{\partial w_i} = \frac{1}{N}\sum_{n=1}^{N}(y_n - t_n)\,x_{ni}$$

Other activation functions *(NE)*

- Tanh



$$g(a) = \tanh(a) = \frac{1-e^{-2a}}{1+e^{-2a}}$$

  - Mapping $(-\infty, +\infty) \rightarrow (-1, 1)$
  - 0 (zero) centred $\rightarrow$ faster convergence than sigmoid

- ReLU (Rectified Linear Unit)

$$g(a) = \max(0, a)$$

  - Several times faster than tanh.
  - 'Dying ReLU' problem – a unit of outputting 0 always

Exercise

1. Show networks with linear nodes in all hidden layers reduce to single-layer networks.
2. Prove that the derivative of the logistic sigmoid function $g(a)$ is given as $g'(a) = g(a)\,(1-g(a))$, and sketch the graph of it.
3. Explain about the learning rate η for the gradient descent method.
4. Explain the problem with the training of a neural network with the MSE criterion when the sigmoid function is used as the activation function.
5. *(NE)* Prove that the partial derivative of the cross-entropy error is given as

$$\frac{\partial E_{\mathrm{H}}(\boldsymbol{w})}{\partial w_i} = \frac{1}{N}\sum_{n=1}^{N}(y_n - t_n)\,x_{ni}\,.$$

Summary

- Limitations of Perceptron
- Solutions to the problems
- Neural network with differentiable non-linear functions (e.g. logistic sigmoid function)
- Training of the network with the gradient descent algorithm
- Considered only a single-layer network with a single-output node
- A very good reference: `http://neuralnetworksanddeeplearning.com/`