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**

  \[ a(\dot{x}) = \mathbf{w}^T \mathbf{x} + w_0 = \dot{\mathbf{w}}^T \dot{x} \]

  where \( \dot{\mathbf{w}} = (w_0, \mathbf{w}^T)^T \), \( \dot{x} = (1, \mathbf{x}^T)^T \)

  \[ x_0 = 1 \]

  \[ y(\dot{x}) = g( a(\dot{x}) ) = g( \dot{\mathbf{w}}^T \dot{x} ) \]

  where \( g(a) = \begin{cases} 
    1, & \text{if } a \geq 0, \\
    0, & \text{if } a < 0
  \end{cases} \)

  \( g(a) \): activation/transfer function

\[
x_2 \geq x_1 - 1 \\
a(x) = 1 - x_1 + x_2 = w_0 + w_1 x_1 + w_2 x_2 \\
w_0 = 1, w_1 = -1, w_2 = 1
\]
Geometry of Perceptron’s error correction

\[ y(x_i) = g(w^T x_i) \]

\[ w^{(\text{new})} \leftarrow w + \eta (t_i - y(x_i)) x_i \quad (0 < \eta < 1) \]

<table>
<thead>
<tr>
<th>( t_i - y(x_i) )</th>
<th>( y(x_i) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

\[ w^T x = \|w\| \|x\| \cos \theta \]
Geometry of Perceptron’s error correction (cont.)

\[ y(x_i) = g(w^T x_i) \]

\[ w^{(\text{new})} \leftarrow w + \eta (t_i - y(x_i)) x_i \quad (0 < \eta < 1) \]

<table>
<thead>
<tr>
<th>( t_i - y(x_i) )</th>
<th>( y(x_i) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

\[ w^T x = \|w\| \|x\| \cos \theta \]
Geometry of Perceptron’s error correction (cont.)

\[ y(x_i) = g(w^T x_i) \]

\[ w^{(\text{new})} \leftarrow w + \eta (t_i - y(x_i)) x_i \quad (0 < \eta < 1) \]

<table>
<thead>
<tr>
<th>( t_i - y(x_i) )</th>
<th>( y(x_i) )</th>
<th>( t_i )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

\[ w^T x = \| w \| \| x \| \cos \theta \]

\( C_0 \)

\( C_1 \)
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 $w$ are adjusted for misclassified data only (correctly classified data are not considered at all)
A limitation of Perceptron

\[ y = g(\mathbf{w}^T \mathbf{x}) \]

\[ z_1 = g(\mathbf{w}_1^{(1)^T} \mathbf{x}) = g(w_{11}^{(1)} x_1 + w_{12}^{(1)} x_2 + w_{10}^{(1)}) \]

\[ z_2 = g(\mathbf{w}_2^{(1)^T} \mathbf{x}) = g(w_{21}^{(1)} x_1 + w_{22}^{(1)} x_2 + w_{20}^{(1)}) \]

\[ y = g(\mathbf{w}^{(2)^T} \mathbf{z}) = g(w_{11}^{(2)} z_1 + w_{12}^{(2)} z_2 + w_{10}^{(2)}) \]
Choices of decision boundaries

(a)

(b)

(c)
How can we resolve the problem of training?

- Use the least squares error criterion for training
  \[ E_2(w) = \sum_{n=1}^{N} (y_n - t_n)^2 \]
- Replace \( g(.) \) with a differentiable function

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

\[ z_i = g(w_i^{(1)}^T x) \quad \Rightarrow \quad z_i = w_i^{(1)}^T x \]

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

Inf2b - Learning: Lectures 12,13  |  Single layer Neural Networks (2,3)
How can we resolve the problem of training? (cont.)

- Replace $g(\cdot)$ with a differentiable non-linear function
  
  e.g., Logistic sigmoid function:
  
  $$g(a) = \frac{1}{1 + e^{-a}} = \frac{1}{1 + \exp(-a)}$$

  
  ![Graph of g(a) = 1 / (1 + exp(-a))](image)

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

  $$\frac{d}{da} g(a) = g'(a) = g(a)(1 - g(a))$$
Assume a single-layer neural network with a single output node with a logistic sigmoid function:

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

\[ g(a) = \frac{1}{1 + \exp(-a)} \]
Training set: \( \mathcal{D} = \{ (x_1, t_1), \ldots, (x_N, t_N) \} \)

where \( t_i \in \{0, 1\} \)

Error function:

\[
E(w) = \frac{1}{2} \sum_{n=1}^{N} (y_n - t_n)^2
\]

\[
= \frac{1}{2} \sum_{n=1}^{N} \left( g(w^T x_n) - t_n \right)^2
\]

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

Definition of the training problem as an optimisation problem

\[
\min_w E(w)
\]
Training of single layer neural network

- Optimisation problem: \( \min_w E(w) \)
- 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.)
  \[
  w_i^{(\text{new})} \leftarrow w_i - \eta \frac{\partial}{\partial w_i} E(w), \quad (\eta > 0)
  \]
  (vector rep.)
  \[
  w^{(\text{new})} \leftarrow w - \eta \nabla_w E(w), \quad (\eta > 0)
  \]
- Online/stochastic gradient descent (cf. Batch training)
  Update the weights one pattern at a time. (See Note 11)
Gradient descent

\[ w_{i}^{(\text{new})} \leftarrow w_{i} - \eta \frac{\partial}{\partial w_{i}} E(w), \quad (\eta > 0) \]
Local minimum problem with the gradient descent

\[ w_i^{(\text{new})} \leftarrow w_i - \eta \frac{\partial}{\partial w_i} E(w), \quad (\eta > 0) \]
Training of the single-layer neural network

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

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

\[
\frac{\partial E(\mathbf{w})}{\partial w_i} = \frac{\partial E(\mathbf{w})}{\partial y_n} \frac{\partial y_n}{\partial a_n} \frac{\partial a_n}{\partial w_i} \\
= \sum_{n=1}^{N} (y_n - t_n) \frac{\partial g(a_n)}{\partial a_n} \frac{\partial a_n}{\partial w_i} \\
= \sum_{n=1}^{N} (y_n - t_n) g'(a_n) x_{ni} \\
= \sum_{n=1}^{N} (y_n - t_n) g(a_n) (1 - g(a_n)) x_{ni} \]
Another training criterion – cross-entropy error

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

\[ E_{\text{MSE}}(w) = \frac{1}{2} \sum_{n=1}^{N} (y_n - t_n)^2, \quad y_n = g(a_n) \]

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

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

- Cross-entropy error \((NE)\)

\[ E_H(w) = -\frac{1}{N} \sum_{n=1}^{N} \left\{ t_n \ln y_n + (1 - t_n) \ln (1 - y_n) \right\} \]

It can be shown that:

\[ \frac{\partial E_H(w)}{\partial w_i} = \frac{1}{N} \sum_{n=1}^{N} (y_n - t_n) x_{ni} \]
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 $\eta$ 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

$$
\frac{\partial E_H(w)}{\partial w_i} = \frac{1}{N} \sum_{n=1}^{N} (y_n - t_n) x_{ni} .
$$
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/