Inf2b Learning and Data
Lecture 15: Multi-layer neural networks (2)

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/spring2017/inf08009learning
Office hours: Wednesdays at 14:00-15:00 in IF-3.04 → 2.46
Jan-Mar 2017

Today’s Schedule
1 Training of neural networks (recap)
2 Activation functions
3 Experimental comparison of different classifiers
4 Overfitting and generalisation
5 Deep Neural Networks

Training of neural networks (recap)

Optimisation problem (training):
\[ \min_w E(w) = \min_w \frac{1}{2} \sum_{n=1}^{N} \| y_n - t_n \|^2 \]

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
  \[ w_i^{\text{new}} \leftarrow w_i - \eta \frac{\partial}{\partial w_i} E(w), \quad (\eta > 0) \]

Overfitting and generalisation

The output can be regarded as a posterior probability.

Logistic sigmoid vs a linear output node

\[ g(a) = \frac{1}{1 + \exp(-a)} \quad \text{vs} \quad g(a) = a \]

No, in terms of classification.
We can replace it with \( g(a) = a \). However, decision boundaries can be different. (NB: A linear decision boundary (\( a = 0 \)) is formed in either case.)

- What benefits are there in using the logistic sigmoid function in the case above?
  - The output can be regarded as a posterior probability.
  - Compared with a linear output node (\( g(a) = a \)), ‘logistic regression’ normally forms a more robust decision boundary against noise.

Logistic classification problem with the least squares error (LSE):

\[ g(a) = \frac{1}{1 + \exp(-a)} \quad \text{vs} \quad g(a) = a \]

[after Fig 4.4b in PRML C. M. Bishop (2006)]
Implementations of gradient descent

\[ E(w) = \frac{1}{2} \sum_{n=1}^{N} || y_n - t_n ||^2 = \frac{1}{2} \sum_{n=1}^{N} \sum_{k=1}^{K} (y_{nk} - t_{nk})^2 \]

- Batch gradient descent:
  \[ w_k \leftarrow w_k - \eta \frac{\partial E}{\partial w_k} \]
- Incremental (online) gradient descent:
  Update weights for each \( x_n \):
  \[ w_k \leftarrow w_k - \eta \frac{\partial E_n}{\partial w_k} \]
- Stochastic gradient descent: c.f. Batch/Mini-batch training
  Update weights for randomly chosen \( x \).

Experimental comparison

- Task: spoken vowel classification
- Classifiers:
  - Gaussian classifier
  - Single layer network (SLN)
  - Multi-layer perceptron (MLP)

Results

- Gaussian classifier: 86.5% correct
- Single layer network: 85.5% correct
- MLP: 86.5% correct

Details of the classifiers

- Gaussian classifier: (2-dimensional) Gaussian for each class. Training involves estimating mean vector and covariance matrix for each class, assume equal priors. (50 parameters)
- Single layer network: 2 inputs, 10 outputs. Iterative training of weight matrix. (30 parameters)
- MLP: two inputs, 25 hidden units, 10 outputs. Trained by gradient descent (backprop). (335 parameters)

For SLN and MLP normalise feature vectors to mean=0 and sd=1:

\[ x_{ni} = \frac{x_{ni} - m_i}{s_i} \]

\( m_i \) is sample mean of feature \( i \) computed from the training set, \( s_i \) is standard deviation.

Decision Regions

- Gaussian classifier
- Single-layer perceptron
- Multi-layer perceptron
### Problems with multi-layer neural networks

- Still difficult to train
  - Computationally very expensive (e.g. weeks of training)
  - Slow convergence ('vanishing gradients')
  - Difficult to find the optimal network topology
- Poor generalisation (under some conditions)
  - Very good performance on the training set
  - Poor performance on the test set

### Overfitting and generalisation

**Example of curve fitting by a polynomial function:**

\[ y(x, w) = w_0 + w_1 x + w_2 x^2 + \ldots + w_M x^M = \sum_{k=0}^{M} w_k x^k \]

(after Fig 1.4 in PRML C. M. Bishop (2006))

- cf. memorising the training data

### Generalisation in neural networks

- **How many hidden units (or, how many weights) do we need?**
  - Optimising training set performance does not necessarily optimise test set performance
    - Network too flexible: Too many weights compared with the number of training examples
    - Network not flexible enough: Not enough weights (hidden units) to represent the desired mapping
- **Generalisation Error:** The predicted error on unseen data. How can the generalisation error be estimated?
  - **Training error?**
    \[ E_{\text{train}} = \frac{1}{M} \sum_{n=1}^{M} (y_n - t_n)^2 \]
  - **Cross-validation error?**
    \[ E_{\text{valid}} = \frac{1}{M} \sum_{n=1}^{M} (y_n - t_n)^2 \]

### Overtraining in neural networks

- **Overtraining** (overfitting) corresponds to a network function too closely fit to the training set (too much flexibility)
- **Undertraining** corresponds to a network function not well fit to the training set (too little flexibility)
- **Solutions**
  - If possible increasing both network complexity in line with the training set size
  - Use prior information to constrain the network function
  - Control the flexibility: **Structural Stabilisation**
  - Control the effective flexibility: **early stopping** and **regularisation**

### Early stopping

- Use validation set to decide when to stop training
- **Training-set error** monotonically decreases as training progresses
- **Validation-set error** will reach a minimum then start to increase
- Effective Flexibility increases as training progresses
- Network has an increasing number of effective degrees of freedom as training progresses
- Network weights become more tuned to training data
- Very effective used in many practical applications such as speech recognition and optical character recognition

### Regularisation — Penalising complexity

- **Original error function**
  \[ E(w) = \frac{1}{2} \sum_{n=1}^{N} ||y_n - t_n||^2 \]
- **Regularised error function**
  \[ \hat{E}(w) = \frac{1}{2} \sum_{n=1}^{N} ||y_n - t_n||^2 + \frac{\lambda}{2} \sum_{j} ||w_j||^2 \]

### Problems with multi-layer neural networks

- Still difficult to train
  - Computationally very expensive (e.g. weeks of training)
  - Slow convergence ('vanishing gradients')
  - Difficult to find the optimal network topology
- Poor generalisation (under some conditions)
  - Very good performance on the training set
  - Poor performance on the test set

### Breakthrough

- 1957 Frank Rosenblatt: 'Perceptron'
- 1986 D. Rumelhart, G. Hinton, and R. Williams: 'Backpropagation'
- 2006 G. Hinton et al (U. Toronto)
  - "Reducing the dimensionality of data with neural networks", Science.
- 2009 J. Schmidhuber (Swiss AI Lab IDSIA)
  - Winner at ICDAR2009 handwriting recognition competition
- 2011- many papers from U. Toronto, Microsoft, IBM, Google, ...
- **What’s the ideas?**
  - **Pretraining**
    - A single layer of feature detectors \( \rightarrow \) Stack it to form several hidden layers
  - **Fine-tuning, dropout**
  - **GPU**
  - **Convolutional network (CNN), Long short-term memory (LSTM)**
  - **Rectified linear unit (ReLU)**
Breakthrough

Speaker-independent phonetic recognition on TIMIT

<table>
<thead>
<tr>
<th>Year</th>
<th>Phone error rate [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1990</td>
<td>20</td>
</tr>
<tr>
<td>1995</td>
<td>22</td>
</tr>
<tr>
<td>2000</td>
<td>24</td>
</tr>
<tr>
<td>2005</td>
<td>26</td>
</tr>
<tr>
<td>2010</td>
<td>28</td>
</tr>
<tr>
<td>2015</td>
<td>30</td>
</tr>
</tbody>
</table>

Summary

- Error back propagation training
- Logistic sigmoid vs linear node
- Decision boundaries
- Overfitting vs generalisation
- (Feed-forward network vs RNN)