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

Jan-Mar 2015
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
Optimisation problem (training):

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

No analytic solution (no closed form)

Employ an iterative method (requires initial values)

- Gradient descent (steepest descent), Newton’s method, Conjugate gradient methods

Gradient descent

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

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

where \( a_n = \sum_{i=0}^{d} w_i x_{ni} \). \( \frac{\partial a_n}{\partial w_i} = x_{ni} \)

\[ \frac{\partial E(w)}{\partial w_i} = \frac{\partial E(w)}{\partial y_n} \frac{\partial y_n}{\partial a_n} \frac{\partial a_n}{\partial w_i} \]

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

\[ = \sum_{n=1}^{N} (y_n - t_n) g'(a_n) x_{ni} \]
Multi-layer neural networks (MLP)

- **Hidden-to-output weights:**
  
  \[
  w_{kj}^{(2)} \leftarrow w_{kj}^{(2)} - \eta \frac{\partial E}{\partial w_{kj}^{(2)}}
  \]

- **Input-to-hidden weights:**
  
  \[
  w_{ji}^{(1)} \leftarrow w_{ji}^{(1)} - \eta \frac{\partial E}{\partial w_{ji}^{(1)}}
  \]
The derivatives of the error function (two-layers) (recap)

\[ E_n = \frac{1}{2} \sum_{k=1}^{K} (y_{nk} - t_{nk})^2 \]

\[ y_{nk} = g(a_{nk}), \quad a_{nk} = \sum_{j=1}^{M} w_{kj}^{(2)} z_{nj} \]

\[ z_{nj} = h(b_{nj}), \quad b_{nj} = \sum_{i=0}^{d} w_{ji}^{(1)} x_{ni} \]

\[ \frac{\partial E_n}{\partial w_{kj}^{(2)}} = \frac{\partial E_n}{\partial y_{nk}} \frac{\partial y_{nk}}{\partial a_{nk}} \frac{\partial a_{nk}}{\partial w_{kj}^{(2)}} = (y_{nk} - t_{nk}) g'(a_{nk}) z_{nj} \]

\[ \frac{\partial E_n}{\partial w_{ji}^{(1)}} = \frac{\partial E_n}{\partial z_{nj}} \frac{\partial z_{nj}}{\partial b_{nj}} \frac{\partial b_{nj}}{\partial w_{ji}^{(1)}} = \left( \sum_{k=1}^{K} (y_{nk} - t_{nk}) \frac{\partial y_{nk}}{\partial z_{nj}} \right) h'(b_{nj}) x_{ni} \]

\[ = \left( \sum_{k=1}^{K} (y_{nk} - t_{nk}) g'(a_{nk}) w_{kj}^{(2)} \right) h'(b_{nj}) x_{ni} \]
\[
\frac{\partial E_n}{\partial w_{kj}^{(2)}} = \frac{\partial E_n}{\partial y_{nk}} \frac{\partial y_{nk}}{\partial a_{nk}} \frac{\partial a_{nk}}{\partial w_{kj}^{(2)}} \\
= (y_{nk} - t_{nk}) g'(a_{nk}) z_{nj} \\
= \delta_{nk}^{(2)} z_{nj}, \quad \delta_{nk}^{(2)} = \frac{\partial E_n}{\partial a_{nk}}
\]

\[
\frac{\partial E_n}{\partial w_{ji}^{(1)}} = \frac{\partial E_n}{\partial z_{nj}} \frac{\partial z_{nj}}{\partial b_{nj}} \frac{\partial b_{nj}}{\partial w_{ji}^{(1)}} \\
= \left( \sum_{k=1}^{K} (y_{nk} - t_{nk}) g'(a_{nk}) w_{kj}^{(2)} \right) h'(b_{nj}) x_{ni} \\
= \left( \sum_{k=1}^{K} \delta_{nk}^{(2)} w_{kj}^{(2)} \right) h'(b_{nj}) x_{ni}
\]
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 \( g(a) = a \). However, decision boundaries can be different. (NB: A linear decision boundary \( a = 0.5 \) 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.
Binary 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))
Different implementations of gradient descent

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

\[ = \sum_{n=1}^{N} E_n, \quad \text{where} \quad E_n = \frac{1}{2} \sum_{k=1}^{K} (y_{nk} - t_{nk})^2 \]

- **Batch gradient descent:**
  \[ w_{ki} \leftarrow w_{ki} - \eta \frac{\partial E}{\partial w_{ki}} \]

- **Incremental (online) gradient descent:**
  Update weights for each \( x_n \)
  \[ w_{ki} \leftarrow w_{ki} - \eta \frac{\partial E_n}{\partial w_{ki}} \]

- **Stochastic gradient descent:**
  Update weights for randomly chosen \( x \).
Experimental comparison

- Task: spoken vowel classification

- Classifiers:
  - Gaussian classifier
  - Single layer network (SLN)
  - Multi-layer perceptron (MLP)
Gaussian for each class

Peterson–Barney F1–F2 Vowel Training Data

multi-layer neural networks (2)
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:

$$z_{ni} = \frac{x^n_i - m_i}{s_i}$$

$m_i$ is sample mean of feature $i$ computed from the training set, $s_i$ is standard deviation.
Results

Gaussian classifier: 86.5% correct
Single layer network: 85.5% correct
MLP: 86.5% correct
Decision Regions: Gaussian classifier

Peterson–Barney F1–F2 Gaussian Decision Regions
Decision Regions: Single-layer perceptron

PetersonBarney F1F2 SLN Decision Regions

F1/Hz
F2 / Hz
Obstacles to 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
Example of curve fitting by a polynomial function:

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

(\textit{after Fig 1.4 in PRML C. M. Bishop (2006)})

- cf. memorising the training data
Overfitting and generalisation

Data set

- training
- validation
- test

Recognition accuracy

# parameters

# training samples

# parameters

Reliability

Model complexity

(generalization)

Accuracy

Training set

Test set

Validation

Training

Test

# parameters

Inf2b Learning and Data: Lecture 15

Multi-layer neural networks (2)
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 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?
  
  \[ E_{\text{train}} = \frac{1}{2} \sum_{\text{training set}} \sum_{k=1}^{K} (y_k - t_k)^2 \]

- Cross-validation error?
  
  \[ E_{\text{exval}} = \frac{1}{2} \sum_{\text{validation set}} \sum_{k=1}^{K} (y_k - t_k)^2 \]
Cross-validation

- Optimise network performance given a fixed training set
- Hold out a set of data (validation set) and predict generalisation performance on this set
  1. Train network in usual way on training data
  2. Estimate performance of network on validation set
- If several networks trained on the same data, choose the one that performs best on the validation set (not the training set)
- *k*-fold Cross-validation: divide the data into *k* partitions; select each partition in turn to be the validation set, and train on the remaining (*k* − 1) partitions. Estimate generalisation error by averaging over all validation sets.
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
Early stopping

![Diagram showing the relationship between validation and training error over time. The diagram indicates that validation error decreases sharply with training time, reaching a minimum at \( t^* \) and then increasing as training continues.]
Regularisation — Penalising complexity (†)

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

- Regularised error function
  \[ \tilde{E}(w) = \frac{1}{2} \sum_{n=1}^{N} \| y_n - t_n \|^2 + \frac{\beta}{2} \sum_{\ell} \| w \|^2 \]
Obstacles to 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
1957  Frank Rosenblatt : 'Perceptron'
1986  D. Rumelhart, G. Hinton, and R. Williams: 'Backpropagation'
2006  G. Hinton etal (U. Toronto)
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  →  Stack it to form several hidden layers
  - Fine-tuning
  - GPU
  - Convolutional network
Speaker-independent phonetic recognition on TIMIT

Phone error rate [%]

Year


Phone error rate [%]

Year

Summary

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