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 2014

Today's Schedule

Training of neural networks (recap)

- 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) e.g. Gradient descent (steepest descent), Newton's method, Conjugate gradient methods
- Gradient descent

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

Training of the single-layer neural network (recap)

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

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

$$\frac{\partial E(\mathbf{w})}{\partial w_i} = \frac{\partial E(\mathbf{w})}{\partial y^{(n)}}\frac{\partial y^{(n)}}{\partial a^{(n)}}\frac{\partial a^{(n)}}{\partial w_i}$$

$$= \sum_{n=1}^{N}(y^{(n)} - t^{(n)})\frac{\partial g(a^{(n)})}{\partial a^{(n)}}\frac{\partial a^{(n)}}{\partial w_i}$$

$$= \sum_{n=1}^{N}(y^{(n)} - t^{(n)})\, g'(a^{(n)})\, x_i^{(n)}$$

Multi-layer neural networks (recap)

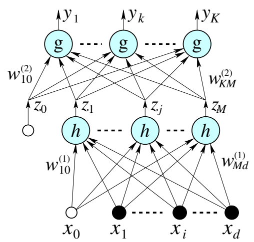
Multi-layer perceptron (MLP)

- Hidden-to-output weights:

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

- Input-to-hidden weights:

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



The derivatives of the error function (two-layers) (recap)

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

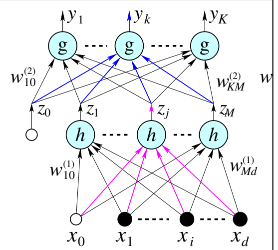
$$y_k^{(n)} = g(a_k^{(n)}), \quad a_k^{(n)} = \sum_{j=1}^{M} w_{kj}^{[2]} z_j^{(n)}$$

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

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

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

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



Error back propagation (recap)

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

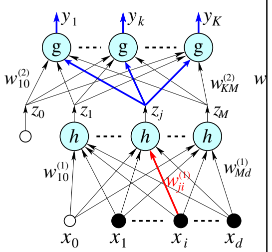
$$= (y_k^{(n)} - t_k^{(n)})\, g'(a_k^{(n)})\, z_j^{(n)}$$

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

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

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

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



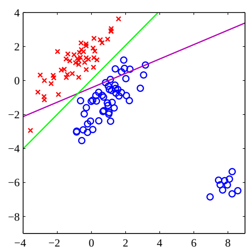
Some questions on activation functions

- 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.

Logistic sigmoid vs a linear output node

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

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

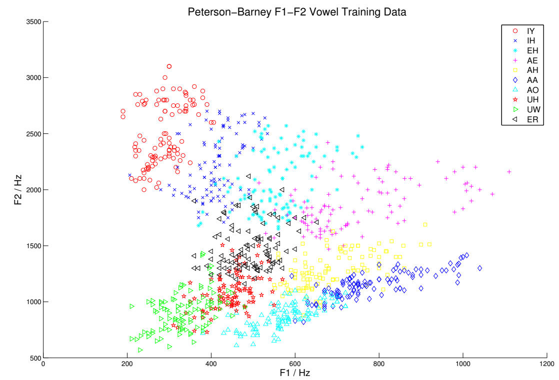
$$= \sum_{n=1}^{N} E^{(n)}, \quad \text{where } E^{(n)} = \frac{1}{2}\sum_{k=1}^{K}(y_k^{(n)} - t_k^{(n)})^2$$

- 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 $\mathbf{x}$.

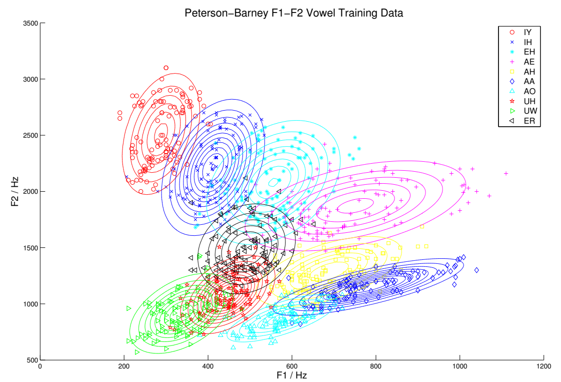
Experimental comparison

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

Classifying spoken vowels (lecture 09) — Training data



Gaussian for each class



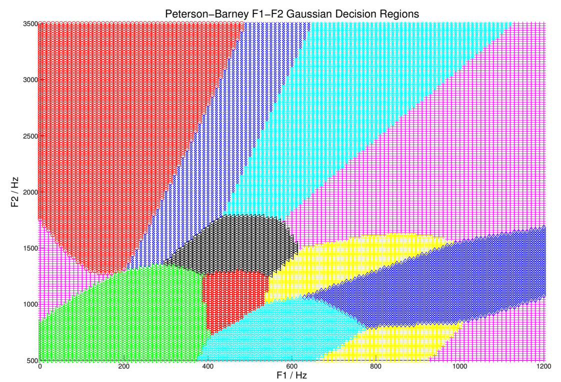
Details of the classifiers

- **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:
  $$z_i^n = \frac{x_i^n - mi}{s_i}$$
  m_i is sample mean of feature i computed from the training set, s_i is standard deviation.

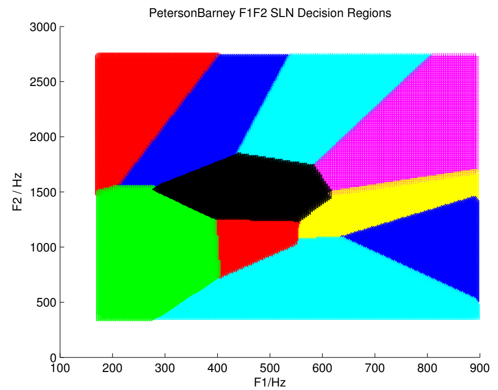
Results

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

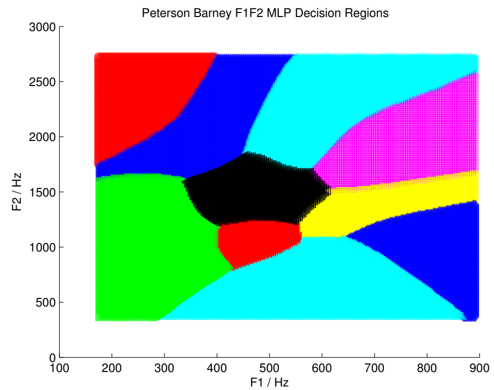
Decision Regions: Gaussian classifier



Decision Regions: Single-layer perceptron



Decision Regions: Multi-layer perceptron



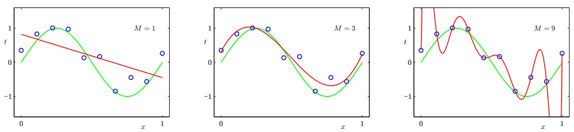
Obstacles to multi-layer neural networks

- 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

Overfitting and generalisation

Example of curve fitting by a polynomial function:

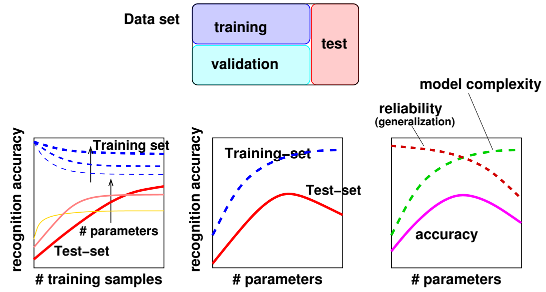
$$y(x, \mathbf{w}) = w_0 + w_1 x + w_2 x^2 + \ldots + w_M x^M = \sum_{k=0}^{M} w_k x^k$$



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

- cf. memorising the training data

Overfitting and generalisation



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?
    $$E_{\text{train}} = \frac{1}{2} \sum_{\text{trainingset}} \sum_{k=1}^{K} (y_k - t_k)^2$$
  - Cross-validation error?
    $$E_{\text{xval}} = \frac{1}{2} \sum_{\text{validationset}} \sum_{k=1}^{K} (y_k - t_k)^2$$

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.

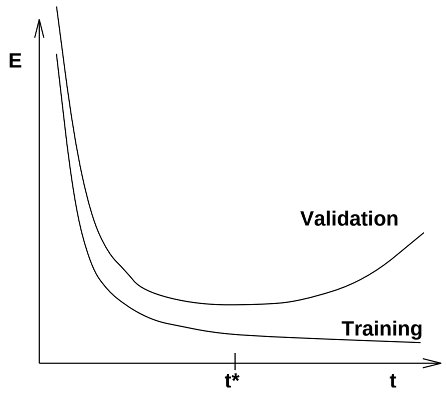
Overtraining in neural networks

- 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 $^{(\dagger)}$

- 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



Regularisation — Penalising complexity $^{(\dagger)}$

- Original error function
  $$E(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^{N} ||\mathbf{y}^{(n)} - \mathbf{t}^{(n)}||^2$$
- Regularised error function
  $$\tilde{E}(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^{N} ||\mathbf{y}^{(n)} - \mathbf{t}^{(n)}||^2 \; + \; \frac{\beta}{2} \sum_{\ell} ||\mathbf{w}||^2$$

Obstacles to multi-layer neural networks

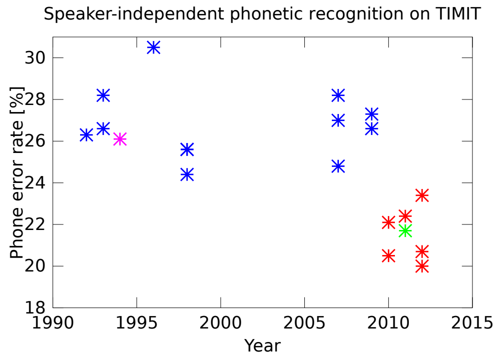
- Still difficult to train
  - Computationally very expensive (e.g. weeks of training)
  - Slow convergence ('vanishing gradients')
  - Difficult to find the optimal network topology
- Poor generalisation (under some conditions)
  - Very good performance on the training set
  - Poor performance on the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

Breakthrough (†)

1957 Frank Rosenblatt : 'Perceptron'
1986 D. Rumelhart, G. Hinton, and R. Williams: 'Backpropagation'
2006 G. Hinton etal (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
  - GPU
  - Convolutional network

Breakthrough (†)



Summary

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