Today’s Schedule

- Training of neural networks (recap)
- Activation functions
- Experimental comparison of different classifiers
- Overfitting and generalisation
- Deep Neural Networks

Training of neural networks (recap)

- Optimisation problem (training):
  \[ \min_w E(w) = \min_w \sum_{n=1}^N \|y_n - t_n\|^2 \]
  - 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
    \[ w_i^{(new)} \leftarrow w_i - \eta \frac{\partial}{\partial w_i} E(w), \quad (\eta > 0) \]

Error back propagation (recap)

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

Multi-layer neural networks (recap)

- Multi-layer perceptron (MLP)
  - Hidden-to-output weights:
    \[ w_k^{(2)} \leftarrow w_k^{(2)} - \eta \frac{\partial E}{\partial w_k} \]
  - Input-to-hidden weights:
    \[ w_i^{(1)} \leftarrow w_i^{(1)} - \eta \frac{\partial E}{\partial w_i} \]

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.)

Logistic sigmoid vs a linear output node

- Binary classification problem with the least squares error (LSE):
  \[ g(a) = \frac{1}{1 + \exp(-a)} \quad \text{vs} \quad g(a) = a \]

(Extended from Lecture 15: Multi-layer neural networks (2))
Different implementations of gradient descent

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

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

- Batch gradient descent:
  $$w_j \leftarrow w_j - \frac{\partial E}{\partial w_{j,i}}$$

- Incremental (online) gradient descent:
  Update weights for each $$x_n$$
  $$w_j \leftarrow w_j - \frac{\partial E_n}{\partial w_{j,i}}$$

- 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)

### 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, 20 outputs. Iterative training of weight matrix. (20 parameters)
- **MLP:** 2 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:
  $$x_{m}^{i} = \frac{x_{i}^{j} - m_{i}}{s_{i}}$$

  where $$m_{i}$$ is sample mean of feature $$i$$ computed from the training set, sd=1, is standard deviation.

### Decision Regions

- **Gaussian classifier:**
- **Single-layer perceptron:**
- **Multi-layer perceptron:**

---

**Peterson–Barney F1–F2 Vowel Training Data**

**Peterson–Barney F1–F2 SLN Decision Regions**

**Peterson–Barney F1–F2 MLP Decision Regions**
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:

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

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

- cf. memorising the training data

Overfitting and generalisation

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
- Train network in usual way on training data
- 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.

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

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
  \[ E(w) = \frac{1}{N} \sum_{n=1}^{N} ||y_n - t_n||^2 \]
- Regularised error function
  \[ E(w) = \frac{1}{N} \sum_{n=1}^{N} ||y_n - t_n||^2 + \frac{\beta}{2} \sum_{l} ||w_l||^2 \]
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 et al (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

Summary

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