Multi-layer neural networks (recap)

Training of neural networks (recap)

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

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

Activation functions

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

Input-to-hidden weights:
\[
w^{(1)}_n \leftarrow w^{(1)}_n - \eta \frac{\partial E}{\partial w^{(1)}_n}
\]

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

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

\[
y_{nk} = g(a_{nk}), \quad a_{nk} = \sum_{j=1}^{W_2} w^{(2)}_{kj} b_{nj} \quad b_{nj} = w^{(1)}_{nj} x_{ni}
\]

Error back propagation (recap)

Some questions on activation functions

Logistic sigmoid vs a linear output node

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

Is the logistic sigmoid function necessary for single-layer single-output-node network?

No. In terms of classification.

We can replace it with \(g(\mathbf{a}) = \mathbf{a}\). 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 \(g(\mathbf{a}) = \mathbf{a}\), ‘logistic regression’ normally forms a more robust decision boundary against noise.
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:
  \[ w_{k} \leftarrow w_{k} - \eta \frac{\partial E_n}{\partial w_{k}} \]

- 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

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.

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

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

$$ E_{\text{train}} = \frac{1}{N} \sum_{n=1}^{N} ||y_n - t_n||^2 $$

Cross-validation error?

$$ E_{\text{valid}} = \frac{1}{K} \sum_{k=1}^{K} \sum_{n=1}^{N_k} ||y_n - t_n||^2 $$

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
  $$ \tilde{E}(w) = \frac{1}{2} \sum_{n=1}^{N} ||y_n - t_n||^2 + \frac{\beta}{2} \sum_{\ell} ||w||^2 $$

Ability of neural networks

- Universal approximation theorem
  - "Univariate function and a set of affine functionals can uniformly approximate any continuous function of n real variables with support in the unit hypercube; only mild conditions are imposed on the univariate function."
  - (G. Cybenko (1989))
  - A single-output node neural network with a single hidden layer with a finite neurons can approximate continuous functions.

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
1957  Frank Rosenblatt: ‘Perceptron’
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
      • Fine-tuning, dropout
      • Convolutional network (CNN), Long short-term memory (LSTM)
      • Rectified linear unit (ReLU)

Speaker-independent phonetic recognition on TIMIT

Summary
- Error back propagation training
- Logistic sigmoid vs linear node
- Decision boundaries
- Overfitting vs generalisation
- (Feed-forward network vs RNN)
- A very good reference: http://neuralnetworksanddeeplearning.com/