Inf2b - Learning
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/spring2020/infr08028
Office hours: Wednesdays at 14:00-15:00 in IF-3.04

Jan-Mar 2020
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):

$$\min_{\mathbf{w}} E(\mathbf{w}) = \min_{\mathbf{w}} \frac{1}{2} \sum_{n=1}^{N} (y_n - t_n)^2$$

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

$$w_i^{(\text{new})} \leftarrow w_i - \eta \frac{\partial}{\partial w_i} E(\mathbf{w}), \quad (\eta > 0)$$
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 \( y_n = g(a_n) \), \( a_n = \sum_{i=0}^{D} w_i x_{ni} \), \( \frac{\partial a_n}{\partial w_i} = x_{ni} \)

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

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

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

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

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

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

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

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

\[
= \left( \sum_{k=1}^{K}(y_{nk} - t_{nk}) g'(a_{nk}) w_{kj}^{(2)} \right) h'(b_{nj}) x_{ni}
\]
Error back propagation (recap)

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

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

\[
= \left(\sum_{k=1}^{\kappa} \delta_{nk}^{(2)} w_{kj}^{(2)}\right) h'(b_{nj}) x_{ni}
\]
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.
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))
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:** 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)
Peterson–Barney F1–F2 Vowel Training Data

IY
IH
EH
AE
AH
AA
AO
UH
UW
ER
Gaussian for each class

Peterson–Barney F1–F2 Vowel Training Data

F1 / Hz
F2 / Hz

IY
IH
EH
AE
AH
AA
AO
UH
UW
ER
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.
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

F1 / Hz
F2 / Hz

Inf2b - Learning: Lecture 15 | Multi-layer neural networks (2)
Decision Regions: Single-layer perceptron

PetersonBarney F1F2 SLN Decision Regions

F1/Hz
F2 / Hz

Inf2b - Learning: Lecture 15  Multi-layer neural networks (2)
Decision Regions: Multi-layer perceptron

Peterson Barney F1F2 MLP Decision Regions

F1 / Hz
F2 / Hz

Inf2b - Learning: Lecture 15
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?
  \[ E_{\text{train}} = \frac{1}{2} \sum_{\text{training set}} \sum_{k=1}^{K} (y_k - t_k)^2 \]

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

The graph illustrates the concept of early stopping in the context of training and validation.

- **Training** and **Validation** error curves are plotted over time.
- The **Validation** curve shows a decrease in error with time, while the **Training** curve also decreases but may continue decreasing even after the validation error starts to increase.
- The point **t** represents the time at which training should ideally be stopped to avoid overfitting.
- The point **t** is preceded by a point **t^***, which is the optimal stopping time to balance training and validation errors.

This diagram helps in understanding how early stopping can prevent overfitting by stopping training before the validation error starts to increase.
Regularisation — Penalising complexity (**(†)**)

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

- **Regularised error function**
  \[
  \tilde{E}(w) = \frac{1}{2} \sum_{n=1}^{N} ||y_n - t_n||^2 + \frac{\beta}{2} \sum_{\ell} ||w||^2
  \]
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.

K. Hornik (1990) doi:10.1016/0893-6080(91)90009-T
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)
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)
Speaker-independent phonetic recognition on TIMIT

Phone error rate [%] vs. Year
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/