Inf2b Learning and Data

Lecture 12 (Chapter 12): Multi-layer neural networks

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/spring2018/infr08009learning
Office hours: Wednesdays at 14:00-15:00 in IF-3.04

Jan-Mar 2018
Today’s Schedule

1. Single-layer network with a single output node (recap)
2. Single-layer network with multiple output nodes
3. Activation functions
4. Multi-layer neural network
5. Overfitting and generalisation
6. Deep Neural Networks
Single-layer network with a single output node (recap)

- **Activation function:**
  
  \[ y = g(a) = g\left(\sum_{i=0}^{D} w_i x_i\right) \]
  
  \[ g(a) = \frac{1}{1 + \exp(-a)} \]

- **Training set:** \( D = \{(x_n, t_n)\}_{n=1}^{N} \)
  - where \( t_n \in \{0, 1\} \)

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

- **Optimisation problem (training):**
  
  \[ \min_w E(w) \]
Training of single layer neural network (recap)

- Optimisation problem: \( \min_w E(w) \)
- 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
  
  (scalar rep.)
  \[
  w_i^{(\text{new})} \leftarrow w_i - \eta \frac{\partial}{\partial w_i} E(w), \quad (\eta > 0)
  \]
  
  (vector rep.)
  \[
  w^{(\text{new})} \leftarrow w - \eta \nabla_w E(w), \quad (\eta > 0)
  \]

- Online/stochastic gradient descent (cf. Batch training)
  Update the weights one pattern at a time. (See Note 11)
Training of the single-layer neural network (cont.)

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

where \( y_n = g(a_n), \quad a_n = \sum_{i=0}^{D} w_i x_{ni}, \quad \frac{\partial a_n}{\partial w_i} = x_{ni} \)

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

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

\[
= \sum_{n=1}^{N} (y_n - t_n) g'(a_n) x_{ni}
\]
Single-layer network with multiple output nodes

- $K$ output nodes: $y_1, \ldots, y_K$.
- For $x_n = (x_{n0}, \ldots, x_{nD})^T$, 

$$y_{nk} = g \left( \sum_{i=0}^{D} w_{ki} x_{ni} \right) = g(a_{nk})$$

$$a_{nk} = \sum_{i=0}^{D} w_{ki} x_{ni}$$
Training set: \( \mathcal{D} = \{(x_1, t_1), \ldots, (x_N, t_N)\} \)

where \( t_n = (t_{n1}, \ldots, t_{nK}) \) and \( t_{nk} \in \{0, 1\} \)

Error function:

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

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

Training by the gradient descent:

\[
w_{ki} \leftarrow w_{ki} - \eta \frac{\partial E}{\partial w_{ki}}, \quad (\eta > 0)
\]
The derivatives of the error function (single-layer)

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

\[ y_{nk} = g(a_{nk}) \]

\[ a_{nk} = \sum_{i=0}^{d} w_{ki} x_{ni} \]

\[ \frac{\partial E_n}{\partial w_{ki}} = \frac{\partial E_n}{\partial y_{nk}} \frac{\partial y_{nk}}{\partial a_{nk}} \frac{\partial a_{nk}}{\partial w_{ki}} = (y_{nk} - t_{nk}) g'(a_{nk}) x_{ni} \]
Notes on Activation functions

Interpretation of output values

Normalisation of the output values
Output of logistic sigmoid activation function

Consider a single-layer network with a single output node logistic sigmoid activation function:

\[ y = g(a) = \frac{1}{1 + \exp(-a)} = g \left( \sum_{i=0}^{D} w_i x_i \right) \]

Consider a two class problem, with classes \( C_1 \) and \( C_2 \). The posterior probability of \( C_1 \):

\[
P(C_1|x) = \frac{p(x|C_1) P(C_1)}{p(x)} = \frac{p(x|C_1) P(C_1)}{p(x|C_1) P(C_1) + p(x|C_2) P(C_2)} = \frac{1}{1 + \exp \left( - \ln \frac{p(x|C_1) P(C_1)}{p(x|C_2) P(C_2)} \right)}
\]
Approximation of posterior probabilities

Logistic sigmoid function

\[ g(a) = \frac{1}{1 + \exp(-a)} \]

Posterior probabilities of two classes with Gaussian distributions:
Normalisation of output nodes

- Outputs with sigmoid activation function:
  \[ \sum_{k=1}^{K} y_k \neq 1 \]
  \[ y_k = g(a_k) = \frac{1}{1 + \exp(-a_k)}, \quad a_k = \sum_{i=0}^{D} w_{ki} x_i \]

- **Softmax** activation function for \( g() \):
  \[ y_k = \frac{\exp(a_k)}{\sum_{\ell=1}^{K} \exp(a_{\ell})} \]

- Properties of the softmax function
  1. \( 0 \leq y_k \leq 1 \)
  2. \( \sum_{k=1}^{K} y_k = 1 \)
  3. Differentiable
  4. \( y_k \approx P(C_k|x) = \frac{p(x|C_k)P(C_k)}{\sum_{\ell=1}^{K} p(x|C_{\ell})P(C_{\ell})} \)
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.

- What benefits are there in using nonlinear activation functions in multi-layer neural networks?
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))
Multi-layer neural networks

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:
  \[ w_{ji}^{(1)} \leftarrow w_{ji}^{(1)} - \eta \frac{\partial E}{\partial w_{ji}^{(1)}} \]
1940s Warren McCulloch and Walter Pitts: ’threshold logic’
    Donald Hebb: ’Hebbian learning’
1957 Frank Rosenblatt: ’Perceptron’
1969 Marvin Minsky and Seymour Papert: limitations of neural networks
1980 Kunihiro Fukushima: ’Neocognitoron’
The derivatives of the error function (two-layers)

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

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

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

\[ \frac{\partial E_n}{\partial w_{kj}^{(2)}} = \frac{\partial E_n}{\partial y_{nk}} \frac{\partial y_{nk}}{\partial a_{nk}} \frac{\partial a_{nk}}{\partial w_{kj}^{(2)}} \]

\[ = (y_{nk} - t_{nk}) g'(a_{nk}) z_{nj} \]

\[ \frac{\partial E_n}{\partial w_{ji}^{(1)}} = \frac{\partial E_n}{\partial z_{nj}} \frac{\partial z_{nj}}{\partial b_{nj}} \frac{\partial b_{nj}}{\partial w_{ji}^{(1)}} \]

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

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

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

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

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

- cf. memorising the training data
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 [%]

Year


Phone error rate [%]

Year


Breakthrough (†)
Summary

- Training of single-layer network

- Activation functions
  - Approximation of posterior probabilities
    - Sigmoid function (for single output node)
    - Softmax function (for multiple output nodes)

- Training of multi-layer network with 'error back propagation'

- A very good reference:
  http://neuralnetworksanddeeplearning.com/