Welcome to the Machine Learning Practical

Deep Neural Networks

Introduction to MLP; Single Layer Networks (1)

Steve Renals and Pavlos Andreadis

Machine Learning Practical — MLP Lecture 1
20 September 2017
http://www.inf.ed.ac.uk/teaching/courses/mlp/

MLP – Course Details

- People
  - Instructors: Steve Renals and Pavlos Andreadis
  - TA: Antreas Antoniou
  - Demonstrators: Todor Danchev, Patric Fulop, Xinnuo Xu
  - (Co-designers: Pawel Swietojanski and Matt Graham)
- Format
  - Assessed by coursework only
  - 1 lecture/week
  - 1 lab/week (but multiple sessions)
    - Signup using the doodle link on the course webpage
    - Labs start next week (week 2)
  - About 9 h/week independent working during each semester
- Online Q&A / Forum – Piazza
  - https://piazza.com/ed.ac.uk/fall2017/infr11132
- **MLP web pages**
  - http://www.inf.ed.ac.uk/teaching/courses/mlp/

Requirements

- **Programming Ability** (we will use python/numpy)
- **Mathematical Confidence**
- **Previous Exposure to Machine Learning** (e.g. Inf2B, IAML)
- **Enthusiasm for Machine Learning**

Do not do MLP if you do not meet the requirements

This course is not an introduction to machine learning

MLP – Course Content

- Main focus: investigating deep neural networks using Python
    - Semester 1: the basics; handwritten digit recognition (MNIST)
      Numpy, Jupyter Notebook
    - Semester 2: project-based, focused on a specific task
      TensorFlow
- Approach: implement DNN training and experimental setups within a provided framework, propose research questions/hypotheses, perform experiments, make some conclusions
- What approaches will you investigate?
    - Single layer networks
    - Multi-layer (deep) networks
    - Convolutional networks
    - Recurrent networks

Practicals and Coursework

- Four pieces of assessed coursework:
  - Semester 1 – using the basic framework from the labs
    1. Basic deep neural networks
       (*due Friday 27 October 2017, worth 10%*)
    2. Experiments on MNIST handwritten digit recognition
       (*due Friday 24 November 2017, worth 25%*)
  - Semester 2 – miniproject using TensorFlow
    3. part 1
       (*due Thursday 15 February 2017, worth 25%*)
    4. part 2
       (*due Friday 23 March 2017, worth 40%*)
- Assessment: coursework assessment will be based on a submitted report

Practical Questions

- *Must I work within the provided framework?* – **Yes**
- *Can I look at other deep neural network software (e.g Theano, PyTorch, ...)?* – **Yes, if you want to**
- *Can I copy other software?* **No**
- *Can I discuss my practical work with other students?* – **Yes**
- *Can we work together?* – **No**

Good Scholarly Practice. Please remember the University requirement as regards all assessed work. Details about this can be found at:

`http://web.inf.ed.ac.uk/infweb/admin/policies/academic-misconduct`

and at:

`http://www.ed.ac.uk/academic-services/staff/discipline`

Reading List

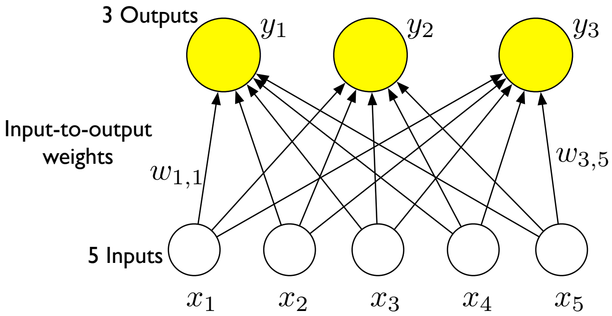
- Yoshua Bengio, Ian Goodfellow and Aaron Courville, *Deep Learning*, 2016, MIT Press.
  http://www.iro.umontreal.ca/~bengioy/dlbook
  https://mitpress.mit.edu/books/deep-learning
  Comprehensive
- Michael Nielsen, *Neural Networks and Deep Learning* 2015.
  http://neuralnetworksanddeeplearning.com
  Introductory
- Christopher M Bishop, *Neural Networks for Pattern Recognition*, 1995, Clarendon Press.
  Old-but-good

Single Layer Networks

Single Layer Networks – Overview

- Learn a system which maps an input vector $\mathbf{x}$ to a an output vector $\mathbf{y}$
- **Runtime:** compute the output $\mathbf{y}$ for each input $\mathbf{x}$
- **Training:** The aim is to optimise the parameters of the system such that the correct $\mathbf{y}$ is computed for each $\mathbf{x}$
- **Generalisation:** We are most interested in the output accuracy of the system for unseen test data
- **Single Layer Network:** Use a single layer of computation (linear) to map between input and output

Single Layer Networks



Single Layer Networks

Input vector $\mathbf{x} = (x_1, x_1, \ldots, x_d)^T$

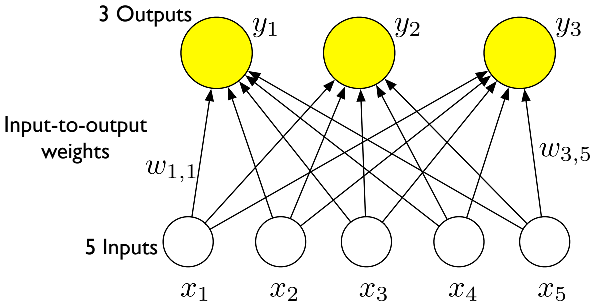
Output vector $\mathbf{y} = (y_1, \ldots, y_K)^T$

Weight matrix $\mathbf{W}$: w_{ki} is the weight from input x_i to output y_k

Bias b_k is the bias for output k

$$y_k = \sum_{i=1}^{d} w_{ki} x_i + b_k \quad ; \quad \mathbf{y} = \mathbf{W}\mathbf{x} + \mathbf{b}$$

Single Layer Networks



$$\mathbf{y} = \mathbf{W}\mathbf{x} + \mathbf{b} \qquad y_k = \sum_{i=1}^{d} W_{ki} x_i + b_k$$

Training Single Layer Networks

Training set N input/output pairs $\{(\mathbf{x}^n, \mathbf{t}^n) : 1 \leq n \leq N\}$

Target vector $\mathbf{t}^n = (t_1^n, \ldots, t_K^n)^T$ – the target output for input $\mathbf{x}^n$

Output vector $\mathbf{y}^n = \mathbf{y}^n(\mathbf{x}^n; \mathbf{W}, \mathbf{b})$ – the output computed by the network for input $\mathbf{x}^n$

Trainable parameters weight matrix $\mathbf{W}$, bias vector $\mathbf{b}$

Training problem Set the values of the weight matrix $\mathbf{W}$ and bias vector $\mathbf{b}$ such that each input $\mathbf{x}^n$ is mapped to its target $\mathbf{t}^n$

Error function Define the training problem in terms of an error function E; training corresponds to setting the weights so as to minimise the error

Supervised learning There is a target output for each input

Error function

- Error function should measure how far an output vector is from its target – e.g. (squared) Euclidean distance – *sum square error*.
  E^n is the error per example:

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

  E is the total error averaged over the training set:

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

- Training process: set **W** and **b** to minimise E given the training set

Weight space and gradients

- **Weight space:** A $K \times d$ dimension space – each possible weight matrix corresponds to a point in weight space. $E(\mathbf{W})$ is the value of the error at a specific point in weight space (given the training data).
- **Gradient** of $E(\mathbf{W})$ given $\mathbf{W}$ is $\nabla_{\mathbf{W}} E$, the matrix of partial derivatives of E with respect to the elements of $\mathbf{W}$:
- **Gradient Descent Training:** adjust the weight matrix by moving a small direction down the gradient, which is the direction along which E decreases most rapidly.
  - update each weight w_{ki} by adding a factor $-\eta \cdot \partial E / \partial w_{ki}$
  - η is a small constant called the *step size* or *learning rate*.
- Adjust bias vector similarly

Gradient Descent Procedure

1. Initialise weights and biases with small random numbers
2. For each epoch (complete pass through the training data)
    1. Initialise total gradients: $\Delta w_{ki} = 0$, $\Delta b_k = 0$
    2. For each training example n:
        1. Compute the error E^n
        2. For all k, i: Compute the gradients $\partial E^n / \partial w_{ki}$, $\partial E^n / \partial b_k$
        3. Update the total gradients by accumulating the gradients for example n

$$\Delta w_{ki} \leftarrow \Delta w_{ki} + \frac{\partial E^n}{\partial w_{ki}} \quad \forall k, i$$

$$\Delta b_k \leftarrow \Delta b_k + \frac{\partial E^n}{\partial b_k} \quad \forall k$$

    3. Update weights:

$$\Delta w_{ki} \leftarrow \Delta w_{ki} / N; \qquad w_{ki} \leftarrow w_{ki} - \eta \Delta w_{ki} \quad \forall k, i$$

$$\Delta b_k \leftarrow \Delta b_k / N; \qquad b_k \leftarrow b_k - \eta \Delta b_k \quad \forall k$$

Terminate either after a fixed number of epochs, or when the error stops decreasing by more than a threshold.

Applying gradient descent to a single-layer network

- Error function:

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

- Gradients:

$$\boxed{\frac{\partial E^n}{\partial w_{rs}}} = \frac{\partial E^n}{\partial y_r}\frac{\partial y_r}{\partial w_{rs}} = \underbrace{(y_r^n - t_r^n)}_{\text{output error}}\ \underbrace{x_s^n}_{\text{input}}$$

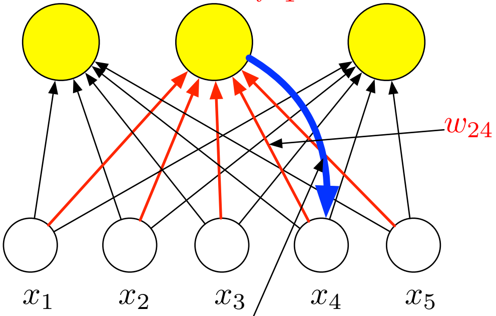
$$\boxed{\frac{\partial E}{\partial w_{rs}}} = \frac{1}{N}\sum_{n=1}^{N}\frac{\partial E^n}{\partial w_{rs}} = \frac{1}{N}\sum_{n=1}^{N}(y_r^n - t_r^n)x_s^n$$

- Weight update

$$w_{rs} \leftarrow w_{rs} - \eta \cdot \frac{1}{N}\sum_{n=1}^{N}(y_r^n - t_r^n)x_s^n$$

Applying gradient descent to a single-layer network

$$y_2 = \sum_{i=1}^{5} w_{2i} x_i$$



$$\Delta w_{24} = \frac{1}{N} \sum_{n=1}^{N} (y_2^n - t_2^n) x_4^n$$

Example: Rainfall Prediction

```
Daily Southern Scotland precipitation (mm). Values may change after QC.
Alexander & Jones (2001, Atmospheric Science Letters).
Format=Year, Month, 1-31 daily precipitation values.
 1931    1   1.40   2.10   2.50   0.10   0.00   0.00   0.90   6.20   1.90   4.90   7.30   0.80   0.30   2
 1931    2   0.90   0.60   0.40   1.10   6.69   3.00   7.59   7.79   7.99   9.59  24.17   1.90   0.20   4
 1931    3                                                                          .10   7.30   6.20   0
 1931    4                                                                          .39   2.40   1.40   1
 1931    5                                                                          .60   1.70  10.83   8
 1931    6                                                                          .30   0.10   9.30  29
 1931    7                                                                          .59   3.90   2.30   7
 1931    8                                                                          .90   1.20   0.50   4
 1931    9                                                                          .23   0.00   1.01   1
 1931   10                                                                          .00   5.10   0.30   0
 1931   11                                                                          .50   0.90   8.50  15
 1931   12                                                                          .10   0.90   2.50   3
 1932    1                                                                          .80  13.71   4.30  17
 1932    2                                                                          .22   0.00   0.00   0
 1932    3   0.10   0.00   0.00   1.60   8.30   4.10  10.00   1.10   0.00   0.00   0.00   0.60   0.50   0
 1932    4   7.41   4.61   1.10   0.10   9.41   8.61   2.10  13.62  17.63   4.71   0.70   0.30  10.02   3
 1932    5   0.10   0.20   0.00   0.10   0.70   0.10   0.80   1.00   0.30   0.00  10.51  17.42   4.11   1
 1932    6   0.00   0.00   0.00   0.20   0.00   0.00   0.60   0.20   0.50   0.00   0.00   0.10   0.00   0
 1932    7   2.41   7.62  13.94   7.42   1.30   1.30   1.80   3.81   2.61   4.01   1.00   4.81   9.93   0
 1932    8   0.00   1.70   0.30   1.00   2.70   4.61   3.40   2.60   0.50   1.30   9.61   1.80   3.81   0
 1932    9  19.37   7.39   9.69   2.70   3.50   3.79  16.68   5.29   4.69  16.88   3.50   1.00  14.08   2
 1932   10   4.40   0.50   0.10   1.80   6.40   8.20  14.69  18.39   4.30   2.80   0.10  16.19   2.20   0
 1932   11  11.37   8.08   5.79   0.00   0.00   0.00   0.00   0.20   0.00   0.00   0.10   0.30   0.00   0
 1932   12  20.23  19.93   3.81   2.40   0.00   0.00   0.00   0.10   0.40   0.40   0.10   0.70   2.30  13
 1933    1   3.40  28.50   2.80  18.80   5.30   4.50  14.60   8.80   0.60   3.50   0.00   3.10   0.50  19
 1933    2   6.10   2.60  14.80  33.10   8.00   9.00   3.10   4.70   7.00   0.10   0.10   0.90   0.10   0
```

How would you train a neural network based on this data?

Do you think it would be an accurate predictor of rainfall?

Exact solution?

A single layer network is a set of linear equations... Can we not solve for the weights diectly given a training set? Why use gradient descent?

This is indeed possible for single-layer systems (consider linear regression!). But direct solutions are not possible for (more interesting) systems with nonlinearities and multiple layers, covered in the rest of the course. So we just look at iterative optimisation schemes.

Summary

- **Reading** – Goodfellow et al, *Deep Learning* chapter 1; sections 4.3 (p79–83), 5.1, 5.7
- Single layer network architecture
- Training sets, error functions, and weight space
- Gradient descent training
- **Lab 1 - next week**: Setup, training data
  Signup using the doodle link on the course webpage!
- **Next lecture:**
  - Stochastic gradient descent and minibatches
  - Classification
  - Introduction to multi-layered networks