Welcome to the Machine Learning Practical

Deep Neural Networks

Introduction to MLP; Single Layer Networks (1)

Steve Renals

Machine Learning Practical — MLP Lecture 1
18 September 2018
http://www.inf.ed.ac.uk/teaching/courses/mlp/

MLP – Course Details

- People
  - Instructors: Hakan Bilen, Steve Renals and Pavlos Andreadis
  - TA: Antreas Antoniou
  - (Co-designers: Pawel Swietojanski and Matt Graham)
- Format
  - Assessed by coursework only
  - 1 lecture/week
  - 1 lab/week (choose one session)
    - Signup at https://doodle.com/poll/gk9xkucg8pgz9369
    - 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/fall2018/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

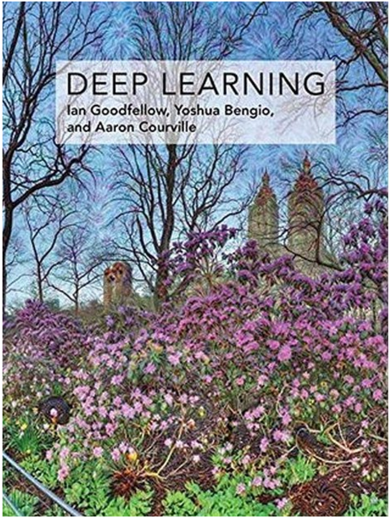
This course is not an introduction to machine learning

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
    Projects groups of 2–3 people
    TensorFlow or PyTorch
- 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

Textbooks

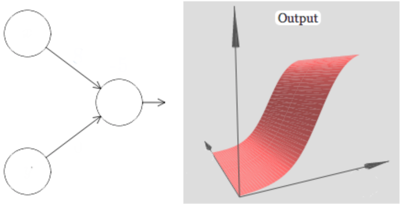
Textbooks


DEEP LEARNING
Ian Goodfellow, Yoshua Bengio,
and Aaron Courville

Textbooks

- Ian Goodfellow, Yoshua Bengio and Aaron Courville, *Deep Learning*, 2016, MIT Press.
  http://www.deeplearningbook.org
  Comprehensive

Textbooks

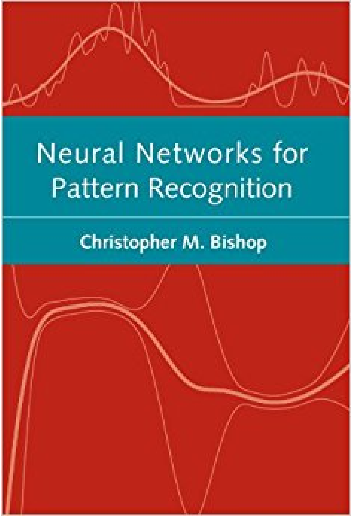


Neural Networks and Deep Learning: A free online book explaining the core ideas behind artificial neural networks and deep learning. Code.

Textbooks

- Ian Goodfellow, Yoshua Bengio and Aaron Courville, *Deep Learning*, 2016, MIT Press.
  http://www.deeplearningbook.org
  Comprehensive
- Michael Nielsen, *Neural Networks and Deep Learning* 2015.
  http://neuralnetworksanddeeplearning.com
  Introductory

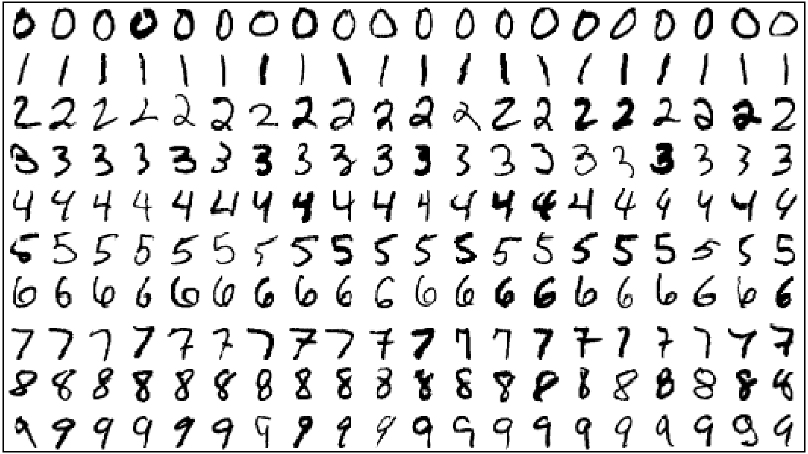
Textbooks


Neural Networks for
Pattern Recognition
Christopher M. Bishop

Textbooks

- Ian Goodfellow, Yoshua Bengio and Aaron Courville, *Deep Learning*, 2016, MIT Press.
  http://www.deeplearningbook.org
  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

MNIST Handwritten Digits



Labs, semester 1

The practical part of MLP is based on a series of labs which explore the material presented in the lectures. The labs are based on the following:

- **Git**: Code and other materials for the labs are available using `git` from a Github repository: https://github.com/CSTR-Edinburgh/mlpractical.
  All necessary git commands will explained as we go along, but if you have not used `git` before, reading a concise guide is helpful, e.g.
  http://rogerdudler.github.io/git-guide/
- **Jupyter notebook**: The labs will be presented as Jupyter notebooks, containing both text and code. The first lab includes an introduction to Jupyter notebook.

Labs, semester 1 (cont)

- **Python/NumPy/Matplotlib**: All the code we use and develop in semester uses Python and the NumPy package. This is briefly introduced in first lab, and if you are new to NumPy we encourage you to go through the tutorial linked from the lab
- **mlp**: A NumPy based neural network package designed specifically for the course that you will (partly) implement and extend during the labs and coursework

As explained in the `README` file on the repository, you need to setup your environment before starting the first lab.

Lab 1: 01_Introduction

After setting up your environment, do the first lab.
The first lab notebook (`01_Introduction.ipynb`) covers:

1. Getting started with **Jupyter Notebook**
2. Introduction to **NumPy** and **Matplotlib** – if you are not familiar with NumPy, then download and follow the Jupyter Notebook tutorial linked from this lab
3. **Data Providers**
   - Modules to load and iterate over data used for training, validating, and testing neural networks
   - `MNISTDataProvider` – class to load and iterate over the MNIST database of handwritten digit images
   - Write your own Data Provider (for the Rainfall (Met Office) data mentioned at the end of this lecture)

(Try to do this by the end of week 2)

Coursework

- Four pieces of assessed coursework:
- Semester 1 – using the basic framework from the labs
  1. Basic deep neural networks, experiments on MNIST (*due Friday 26 October 2018, worth 10%*)
  2. More advanced experiments (*due Friday 23 November 2018, worth 40%*)
- Semester 2 – group project
  3. Interim report (*due Thursday 14 February 2019, feedback only*)
  4. Final report (*due Friday 22 March 2019, worth 50%*)

Practical Questions

- *Must I work within the provided framework in semester 1?*
  - **Yes**
- *Can I look at other deep neural network software?*
  - **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?*
  - **Semester 1: No**
  - **Semester 2: Yes (in groups of 2–3)**

Good scholarly practice – remember the University requirement for assessed work.

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

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

Practical Questions

- *Must I work within the provided framework in semester 1?*
  - **Yes**
- *Can I*
  - **Yes,**
- *Can I*
  - **No**

Any More Questions?

- *Can I discuss my practical work with other students?*
  - **Yes**
- *Can we work together?*
  - **Semester 1: No**
  - **Semester 2: Yes (in groups of 2–3)**

Good scholarly practice – remember the University requirement for assessed work.

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

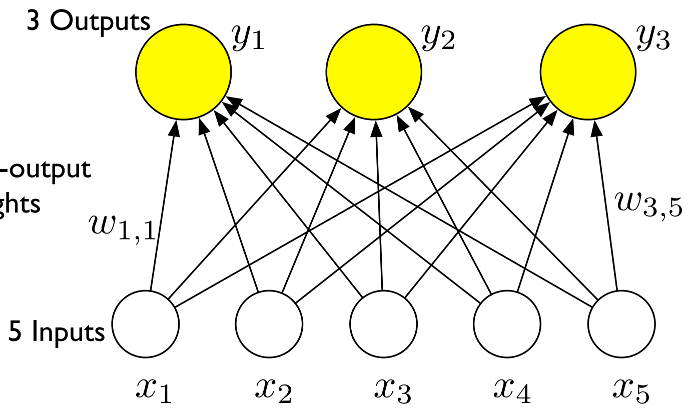
http://www.ed.ac.uk/academic-services/staff/discipline

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:** Optimise the parameters of the network 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 / affine transformation) to map between input and output



3 Outputs
y_1
y_2
y_3
Input-to-output weights
$w_{1,1}$
$w_{3,5}$
5 Inputs
x_1
x_2
x_3
x_4
x_5

Training / Test / Validation Data

Partition the data into training, validation, and test setups

- **Training** set – data used for training the network
- **Validation** set – frequently used to measure the error of a network on "unseen" data (e.g. after each epoch)
- **Test** set – less frequently used "unseen" data, ideally only used once
- Frequent use of the same test data can indirectly "tune" the network to that data (more about this in lecture 5)

Single Layer Networks

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

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

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

Bias b_k is the bias for output k

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

Single Layer Networks

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

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

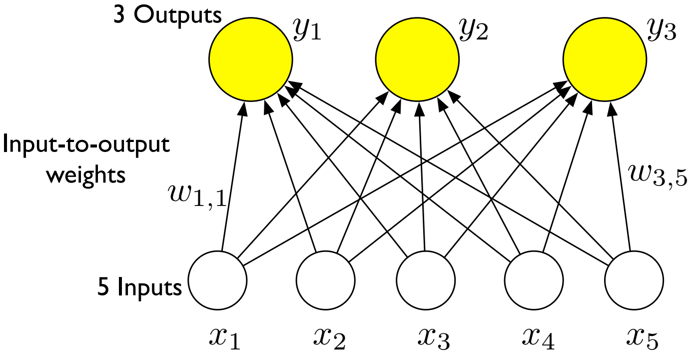
Weight matrix $\boldsymbol{W}$: w_{ki} is the weight from input x_i to output y_k

Bias b_k is the bias for output k

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

Also known as Linear Regression

Single Layer Networks



$$\mathbf{y} = \mathbf{W}\mathbf{x} + \mathbf{b}$$

$$y_k = \sum_{i=1}^{d} W_{ki} x_i + b_k$$

Training Single Layer Networks

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

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

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

Training Single Layer Networks

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

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

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

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

Training Single Layer Networks

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

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

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

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

Supervised learning There is a target output for each input

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

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

Error function

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

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

  E is the total error averaged over the training set:

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

Error function

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

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

  E is the total error averaged over the training set:

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

- Training process: set $\boldsymbol{W}$ and $\boldsymbol{b}$ 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. $E(\boldsymbol{W})$ is the value of the error at a specific point in weight space (given the training data).

Weight space and gradients

- **Weight space:** A $K \times d$ dimension space – each possible weight matrix corresponds to a point in weight space. $E(\boldsymbol{W})$ is the value of the error at a specific point in weight space (given the training data).
- **Gradient** of $E(\boldsymbol{W})$ given $\boldsymbol{W}$ is $\nabla_{\boldsymbol{W}} E$, the matrix of partial derivatives of E with respect to the elements of $\boldsymbol{W}$:

Weight space and gradients

- **Weight space:** A $K \times d$ dimension space – each possible weight matrix corresponds to a point in weight space. $E(\boldsymbol{W})$ is the value of the error at a specific point in weight space (given the training data).
- **Gradient** of $E(\boldsymbol{W})$ given $\boldsymbol{W}$ is $\boldsymbol{\nabla}_{\boldsymbol{W}} E$, the matrix of partial derivatives of E with respect to the elements of $\boldsymbol{W}$:
- **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:

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

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

3. Terminate: after a number of epochs; or when error stops decreasing (threshold).

Applying gradient descent to a single-layer network

- Error function:

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

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:

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

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

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 (`grads_wrt_params`):

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

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

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 (`grads_wrt_params`):

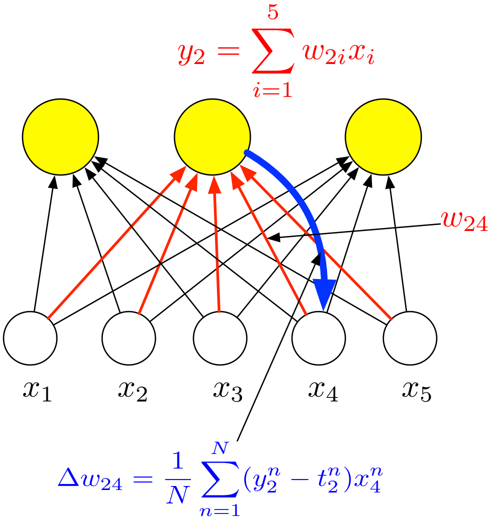
$$\boxed{\frac{\partial E^n}{\partial w_{rs}}} = \underbrace{\frac{\partial E^n}{\partial y_r}}_{\texttt{error\_grad}} \cdot \frac{\partial y_r}{\partial w_{rs}} = \underbrace{(y_r^n - t_r^n)}_{\text{output error}} \cdot \underbrace{x_s^n}_{\text{input}}$$

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

- 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



Lab 2: 02_Single_layer_models

The second lab notebook (`02_Single_layer_models.ipynb`) covers the implementation and training of single-layer networks in NumPy:

1. Efficient implementation of **linear transforms in NumPy** – `numpy.dot` and broadcasting (and timing code using `%%timeit`)
2. Implementing the computations required for **single-layer networks**
    - forward-propagation (`fprop`; y)
    - the error function and its gradient (`error`, `error_grad`; E, $\partial E/\partial y$)
    - gradients with respect to the parameters (`grads_wrt_params`; $\partial E/\partial w_{kj}$)
3. Wrapping it all up into the `mlp` **framework** (`mlp.layers` and `mlp.errors` modules)

(Fine if you don't do this until week 3)

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	2.90	7.50	18.79	1.30	10.
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	4.69	10.58	0.80	0.80	0.
1931	3	0.00	1.30	0.00	0.00	0.00	0.50	0.40	0.60	1.00	0.00	0.10	7.30	6.20	0.20	0.90	0.00	0.00	0.
1931	4	3.99	3.49	0.00	2.70	0.00	0.00	1.80	1.80	0.00	0.20	3.39	2.40	1.40	1.60	3.59	7.99	2.20	0.
1931	5	1.70	0.00	0.70	0.00	5.62	0.70	13.14	0.80	11.13	11.23	0.60	1.70	10.83	8.12	2.21	0.60	0.20	0.
1931	6	1.40	16.40	3.70	0.10	5.80	12.90	4.30	4.50	10.40	13.20	0.30	0.10	9.30	29.60	23.40	2.30	9.80	8.
1931	7	9.49	1.70	8.69	4.10	2.50	13.29	2.70	5.60	3.10	1.30	7.59	3.90	2.30	7.69	1.60	3.60	7.09	1.
1931	8	0.20	0.00	0.00	0.00	0.00	0.60	2.00	0.60	6.60	0.60	0.90	1.20	0.50	4.80	2.80	6.60	4.10	0.
1931	9	9.86	4.33	1.01	0.10	0.30	1.01	0.80	1.31	0.00	0.30	4.23	0.00	1.01	1.01	0.91	14.69	0.40	0.
1931	10	23.18	5.30	4.20	6.89	4.10	11.29	10.09	5.80	11.99	1.80	2.00	5.10	0.30	0.00	0.00	0.10	0.10	0.
1931	11	6.60	20.40	24.80	3.30	3.30	2.60	5.20	4.20	8.00	13.60	3.50	0.90	8.50	15.30	0.10	0.10	13.50	10.
1931	12	3.20	21.60	16.00	5.80	8.40	0.70	6.90	4.80	2.80	1.10	1.10	0.90	2.50	3.20	0.00	0.60	0.10	3.
1932	1	12.71	41.12	22.51	7.20	12.41	5.70	1.70	1.80	24.41	3.80	0.80	13.71	4.30	17.21	20.71	8.50	1.50	1.
1932	2	0.00	0.22	0.00	0.54	0.33	0.11	0.00	0.00	0.22	0.11	0.22	0.00	0.00	0.00	0.00	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	0.00	0.00	0.00	0.00	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	3.61	1.10	0.00	0.00	1.
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	1.00	13.62	0.30	0.10	8.
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	0.10	0.00	0.00	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	0.00	1.20	0.50	0.40	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	0.40	0.70	2.90	0.70	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	2.00	0.40	0.10	0.80	0.
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	0.80	2.40	4.80	20.69	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	0.10	1.30	0.40	0.10	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	13.22	20.43	44.17	27.24	28.
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	19.20	1.10	0.90	0.40	0.
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	0.00	0.20	1.70	0.50	0.
1933	3	2.59	5.29	3.99	5.99	7.19	7.09	0.30	29.54	5.19	0.00	0.00	0.00	1.10	3.89	5.49	2.49	2.89	3.

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   2.90   7.50  18.79   1.30  10.
 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   4.69  10.58   0.80   0.80   0.
 1931   3   0.00   1.30   0.00   0.00   0.00   0.50   0.40   0.60   1.00   0.00   0.10   7.30   6.20   0.20   0.90   0.00   0.00   0.
 1931   4   3.99   3.49   0.00   2.70   0.00   0.00   1.80   1.80   0.00   0.20   3.39   2.40   1.40   1.60   3.59   7.99   2.20   0.
 1931   5   1.7 …                                                                                               2.21   0.60   0.20   0.
 1931   6   1.4 …                                                                                              23.40   2.30   9.80   8.
 1931   7   9.4 …                                                                                               1.60   3.60   7.09   1.
 1931   8   0.2 …                                                                                               2.80   6.60   4.10   0.
 1931   9   9.8 …                                                                                               0.91  14.69   0.40   0.
 1931  10  23.1 …                                                                                               0.00   0.10   0.10   0.
 1931  11   6.6 …                                                                                               0.10   0.10  13.50  10.
 1931  12   3.2 …                                                                                               0.00   0.60   0.10   3.
 1932   1  12.7 …                                                                                              20.71   8.50   1.50   1.
 1932   2   0.0 …                                                                                               0.00   0.00   0.00   0.
 1932   3   0.1 …                                                                                               0.00   0.00   0.00   0.
 1932   4   7.4 …                                                                                               1.10   0.00   0.00   1.
 1932   5   0.1 …                                                                                              13.62   0.30   0.10   8.
 1932   6   0.0 …                                                                                               0.00   0.00   0.00   0.
 1932   7   2.4 …                                                                                               1.20   0.50   0.40   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   0.40   0.70   2.90   0.70   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   2.00   0.40   0.10   0.80   0.
 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   0.80   2.40   4.80  20.69   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   0.10   1.30   0.40   0.10   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  13.22  20.43  44.17  27.24  28.
 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  19.20   1.10   0.90   0.40   0.
 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   0.00   0.20   1.70   0.50   0.
 1933   3   2.59   5.29   3.99   5.99   7.19   7.09   0.30  29.54   5.19   0.00   0.00   0.00   1.10   3.89   5.49   2.49   2.89   3.
```

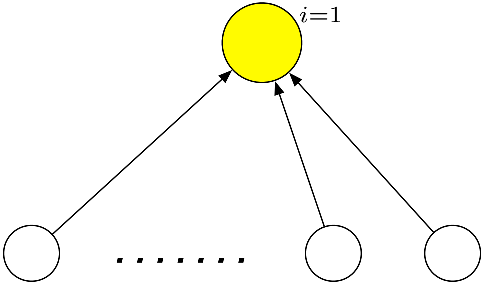
How would you train a neural network based on this data?

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

Single Layer Network for Rainfall Prediction

Output - predicted observation

$$y = \hat{r}_t = \sum_{i=1}^{d} w_i x_i + b$$



$x_1 = r_{t-d}$ .... $x_{d-1} = r_{t-2}$ $x_d = r_{t-1}$

Input - previous d observations

Exact solution?

A single layer network is a set of linear equations... Can we not solve for the weights directly 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: Setup, training data
- Lab 2: Training single-layer networks
- **Signup for labs:** https://doodle.com/poll/gk9xkucg8pgz9369 (One session/week)
- **Office hours:** Tuesdays 16:10-17:00, Appleton Tower Cafe.
- **Next lecture:**
  - Stochastic gradient descent and minibatches
  - Classification
  - Sigmoid and softmax