Welcome to the Machine Learning Practical

This year: Deep Neural Networks

Introduction to MLP; Single Layer Networks

Steve Renals

Machine Learning Practical — MLP Lecture 1
21 September 2016
http://www.inf.ed.ac.uk/teaching/courses/mlp/

MLP – Course Details

- People
  - Lecturer: Steve Renals
  - TA: Matt Graham
  - Demonstrators: Simao Eduardo, Xingxing Zhang
  - Markers: Joachim Fainberg
  - (Co-designer: Pawel Swietojanski)
- Format
  - Assessed by coursework only
  - 1 lecture/week (weeks 1–9, semester 1; weeks 1–5, semester 2)
  - 1 lab/week (but multiple sessions)
  - 20 credits across 2 semesters
- Requirements
  - **Programming Ability** (we will use python/numpy)
  - **Mathematical Confidence**
  - **Knowledge of Machine Learning** (e.g. Inf2B, IAML)

Do not do MLP if you do not meet the requirements

MLP – Course Content

- Main focus: implementing deep neural networks
- Semester 1: the basics; handwritten digit recognition (MNIST)
- Semester 2: focused on a specific task
- Approach: implement DNN training and experimental setups within a provided framework, perform experiments, make some conclusions
- What will you implement?
  - Single layer networks
  - Multi-layer (deep) networks
  - Convolutional networks
  - (Recurrent networks?)

Practicals and Coursework

- Semester 1: practical work will use Python / Numpy / iPython notebook. We'll provide a basic framework (which you will help to write) introduced through the labs.
- Semester 2: practical work will use a toolkit for deep neural networks.
- Four pieces of assessed coursework:
  - Semester 1
    1. Basic deep neural networks, focus on gradient descent optimisation approaches
       (*due Thursday 27 October 2016, worth 10%*)
    2. Experiments on MNIST handwritten digit recognition
       (*due Thursday 24 November 2016, worth 25%*)
  - Semester 2
    3. part 1
       (*due Thursday 2 February 2017, worth 25%*)
    4. part 2
       (*due Thursday 16 March 2017, worth 40%*)

Practical Questions

- *Must I work within the provided framework?* – **Yes**
- *Can I look at other deep neural network software (e.g Theano, Torch, ...)?* – **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://www.ed.ac.uk/academic-services/students/conduct/academic-misconduct
and at:
http://web.inf.ed.ac.uk/infweb/admin/policies/academic-misconduct

Reading List

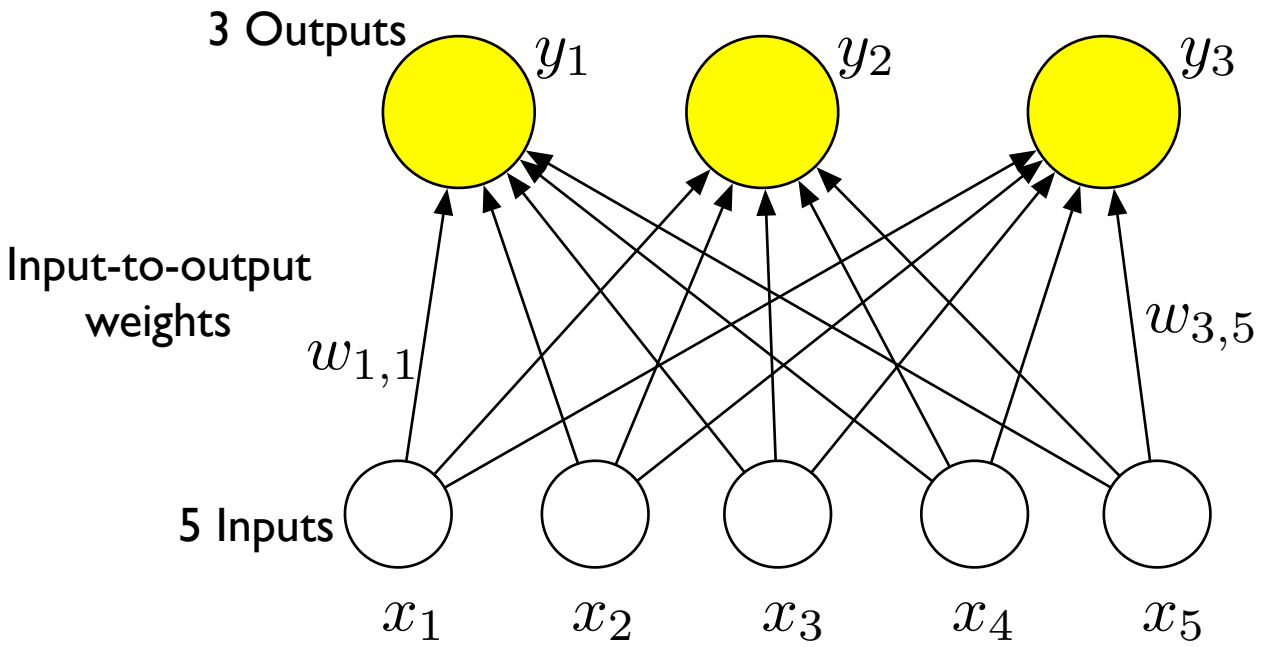
- Michael Nielsen, *Neural Networks and Deep Learning* 2015. http://neuralnetworksanddeeplearning.com
- Yoshua Bengio, Ian Goodfellow and Aaron Courville, *Deep Learning*, 2015. http://www.deeplearningbook.org
- Christopher M Bishop, *Neural Networks for Pattern Recognition*, 1995, Clarendon Press.

Single Layer Networks

Single Layer Networks – Overview

- Learn a system which maps an input vector **x** to a an output vector **y**
- **Runtime:** compute the output **y** for each input **x**
- **Training:** The aim is to optimise the parameters of the system such that the correct **y** is computed for each **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 $\mathbf{x} = (x_1, x_1, \ldots, x_d)^T$

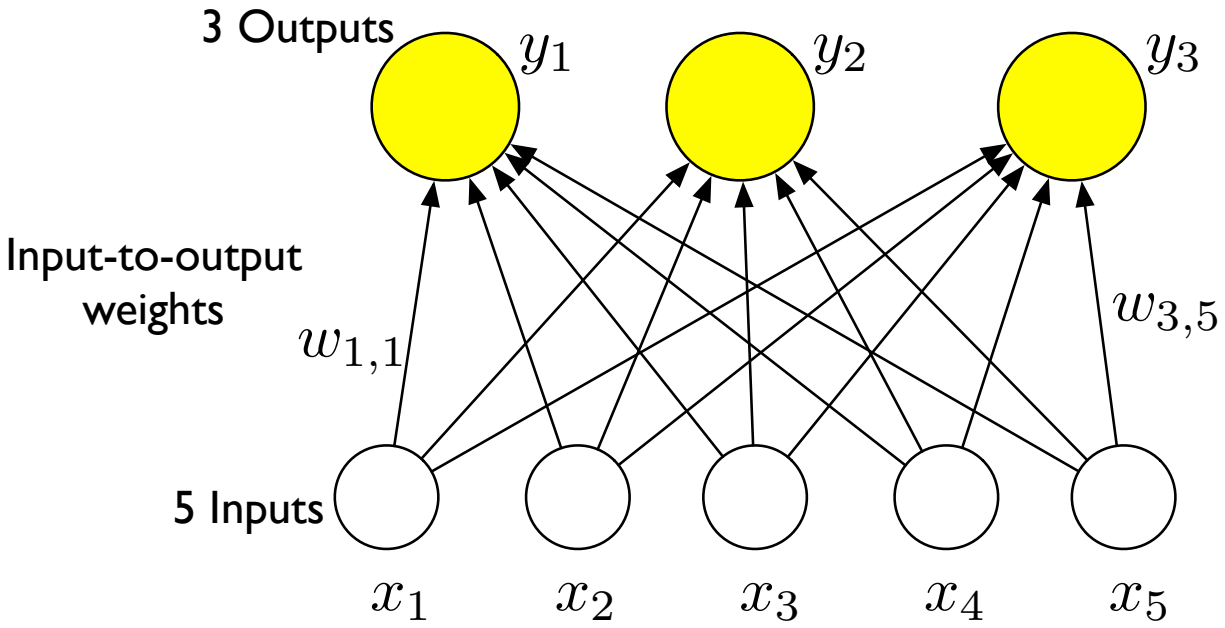
Output vector $\mathbf{y} = (y_1, \ldots, y_K)^T$

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 $\mathbf{x}^n$

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 $E^n(\mathbf{y}^n, \mathbf{t}^n)$. Training corresponds to minimizing the error function $E = \sum_n E^n$

Notes

1. *Supervised* learning – there is a target output for each input.
2. Explicitly show the dependence of $\mathbf{y}^n$ on the weight matrix, bias, and the input vector, by writing $\mathbf{y}^n(\mathbf{x}^n; \mathbf{W}, \mathbf{b})$.

Error function

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

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

$$\boxed{E^n} = \frac{1}{2}||\mathbf{y}^n - \mathbf{t}^n||^2$$

  E is the total error summed over the training set
  E^n is the error for the nth training example

- Can write E^n in terms of components:

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

- Training process: set $\mathbf{W}$ and $\mathbf{b}$ 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 vector of partial derivatives of E with respect to the elements of $\mathbf{W}$:

$$\nabla_{\mathbf{W}} E = \left(\frac{\partial E}{\partial w_{11}}, \ldots, \frac{\partial E}{\partial w_{ki}}, \ldots, \frac{\partial E}{\partial w_{Kd}} \right)^T .$$

- **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:

$$w_{ki} \leftarrow w_{ki} - \eta \Delta w_{ki} \quad \forall k, i$$

$$b_k \leftarrow b_k - \eta \Delta b_k \quad \forall k$$

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

- Need to differentiate the error function with respect to each weight:

$$E^n = \frac{1}{2}\sum_{k=1}^{K}(y_k^n - t_k^n)^2 = \frac{1}{2}\sum_{k=1}^{K}\left(\sum_{i=0}^{d} w_{ki}x_i^n - t_k^n\right)^2$$

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

$$\boxed{\frac{\partial E}{\partial w_{rs}}} = \sum_{n=1}^{N}\frac{\partial E^n}{\partial w_{rs}} = \sum_{n=1}^{N}\delta_r^n x_s^n$$

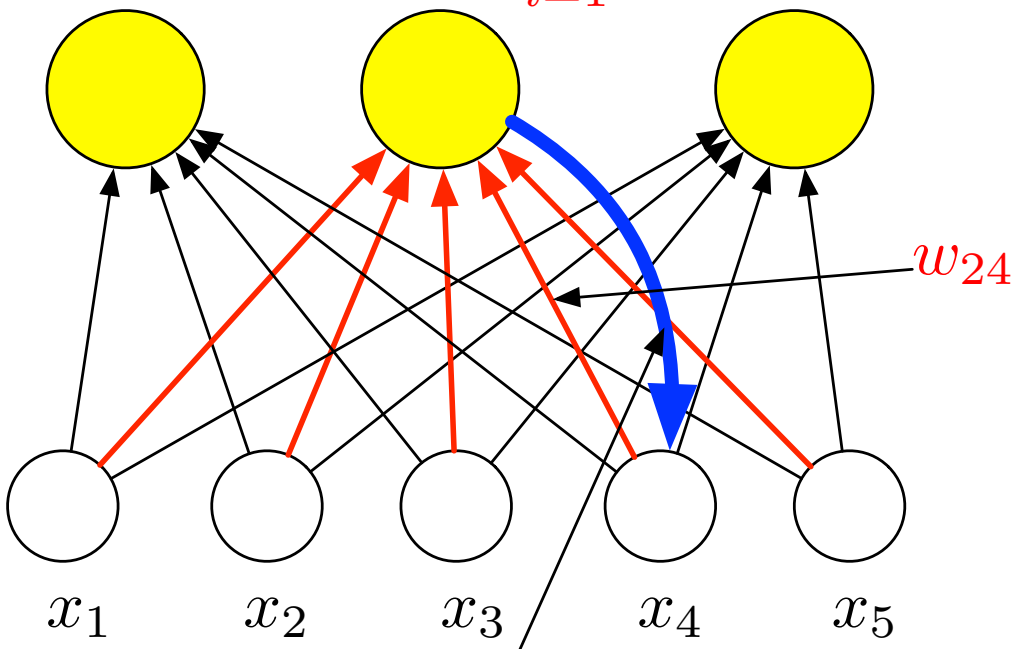
- So the weight update is

$$w_{rs} \leftarrow w_{rs} - \eta\sum_{n=1}^{N}\delta_r^n x_s^n$$

This is sometimes called the *delta rule*.

Applying gradient descent to a single-layer network

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



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

Gradient Descent Pseudocode

procedure GRADIENTDESCENTTRAINING($\mathbf{X}, \mathbf{T}, \mathbf{W}, \mathbf{b}$)
    initialize $\mathbf{W}, \mathbf{b}$ to small random numbers
    while not converged **do**
        for all k, i: $\Delta w_{ki} \leftarrow 0$
        for all k: $\Delta b_k \leftarrow 0$
        for $n \leftarrow 1, N$ **do**
            for $k \leftarrow 1, K$ **do**
                $y_k^n \leftarrow \sum_{i=1}^{d} w_{ki} x_i^n + b_k$
                $\delta_k^n \leftarrow y_k^n - t_k^n$
                for $i \leftarrow 1, d$ **do**
                    $\Delta w_{ki} \leftarrow \Delta w_{ki} + \delta_k^n \cdot x_i^n$
                end for
                $\Delta b_k \leftarrow \Delta b_k + \delta_k^n$
            end for
        end for
        for all k, i: $w_{ki} \leftarrow w_{ki} - \eta \cdot \Delta w_{ki}$
        for all k: $b_k \leftarrow b_k - \eta \cdot \Delta b_k$
    end while
end procedure

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?

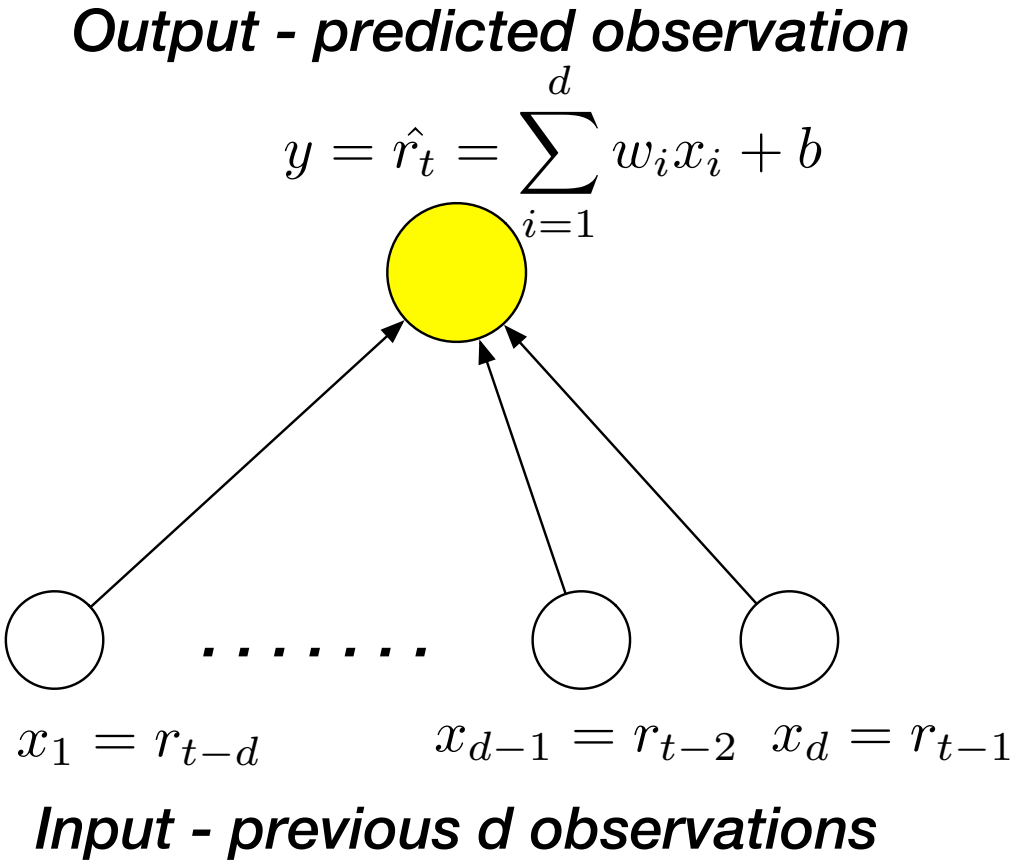
This is indeed possible for single-layer systems (consider linear regression!). But direct solutions are not possible for (more interesting) systems with nonlinearities and multiple layers, covered in the rest of the course. So we just look at iterative optimisation schemes.

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	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
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
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
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
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
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
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
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
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
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
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
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
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
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
1933	4	0.40	14.98	3.20	0.50	0.00	0.00	0.00	11.98	1.70	0.10	4.69	0.20	0.00	0
1933	5	0.00	0.00	4.71	9.92	2.21	13.73	3.81	5.71	1.80	0.10	0.80	0.20	0.00	0

Single Layer Network for Rainfall Prediction



Summary

- Single layer network architecture
- Training sets, error functions, and weight space
- Gradient descent training
- Example – Rainfall prediction
- **Lab 1 - Mon/Tue/Wed 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