TerraWeather
Ensuring Your Safety & Success

Data Modeling Workshop & Challenge

Modeling Weather Radar

1.0 Introduction

Weather radar plays a huge role in making accurate weather forecasts for “smart cities”.

Weather radar works by first emitting a short microwave pulse. This pulse reflects off rain areas at a distance from the emitter, and the backscattered pulse is detected by the radar's receiver. The strength of the backscattered pulse is an indication of the rate of rainfall at the point of backscatter. Figure 1 illustrates this process:

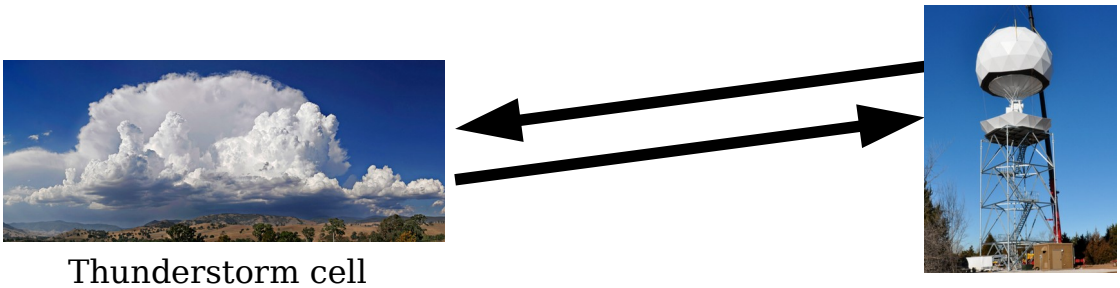


Figure 1: Transmitted and backscattered microwave pulses.
Thunderstorm image courtesy of Fir0002/Flagstaffotos.

The distance of the rain area from the weather radar station is easily estimated from the time between emitting and receiving the pulse. These radar pulses are sent frequently, over a 360° field, so as to cover a large 2D area centered on the radar station.

The backscattered pulse intensity is quantized into 33 “bins”, which should correspond to *rainfall intensity* (mm/hour), for each 300m x 300m “pixel” from the weather radar station.

These are color coded according to an empirical formula to determine the rainfall intensity around the radar station. Figure 2 is an image from a single 360° radar snapshot.

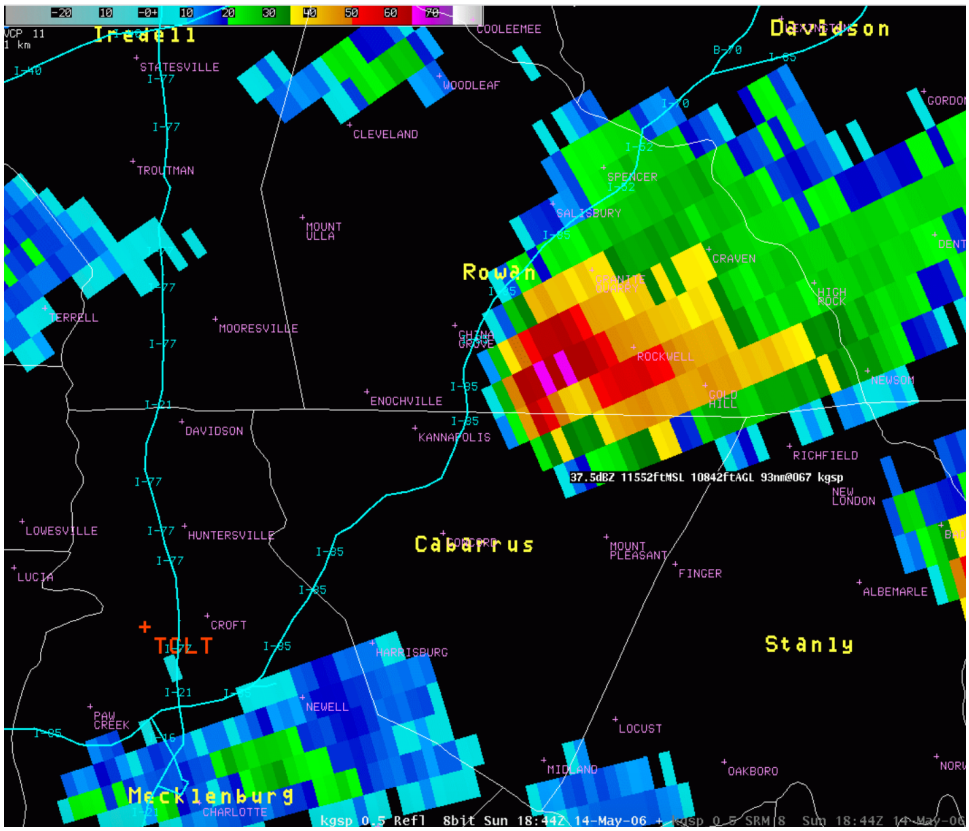


Figure 2: A typical plotted weather radar snapshot. The colored sections correspond to rain areas, with black being rain-free. Note the red thunderstorm cell just off the center of the image.

1.1 A Simple Model

The empirical formula for the rainfall rate (mm/hour) is often in the form:

$$r(x,y) = A e^{Bk} \quad \text{--- (Equation 1)}$$

Where:

- A and B are constants that need to be determined,
- r(x,y) is the rainfall rate at location (x,y) between t and $t+\Delta t$.
- k is the bin of the quantized backscattered signal.

Equation (1) represents a simple *model* of radar rainfall. An accurate determination of the constants A and B can be made by using *rainfall gauge*

data.

Rainfall gauges are fixed facilities (not radar), that continuously collect rainfall data over time. Figure 3 shows a rain gauge:

Figure 3: A rainfall gauge
Image courtesy of famatin

Rain gauge data are reported every 24 hours (1 day), so to make a comparison between radar and gauge rainfall estimates, Equation (1) needs to be amended to:

$$R_{radar}(x,y)=\sum_{k=1}^{33} A e^{Bk} c_k(x,y)\Delta t \quad \text{--- (Equation 2)}$$

Where

- Δt is normalized to hours (eg 5 minutes = 5/60 hours = 0.083 hours)
- the counts $c_k(x,y)$ record the number of times backscatter pulses of quantized intensity *k* has been received over a 24 hour period.
- R(x,y) is now the **total** radar rainfall (ie, in mm) at position (x,y) over a 24-hour period.

1.2 Risk Minimization

To determine A and B, we compare radar rainfall, $R_{radar}(x,y)$ and rain gauge rainfall, $R_{gauge}(x,y)$ adjusting A and B until both are within a desired tolerance

of the other. Listing 1 shows an algorithm called *risk minimization*:

1. Initialize A and B to some values, (eg, some random number or 0)
2. Calculate $R_{rainfall}$ at day t
3. Calculate a *loss,* $L(t)=(R_{radar}-R_{gauge})^2$ at day t
4. Calculate the *risk,* $r=\frac{1}{T}\sum L(t)$ where T is the total period for which we have data.
5. Store the initial risk , r_0
6. **Somehow adjust** A and B so that the new risk $r_1 < r_0$. If this can't be done, stop. If yes, replace r_0 by r_1 and repeat step (6).

Listing 1: Risk Minimization Algorithm

The idea is that as the iterations progress, the successive values of A and B cause the risk (ie, average error) to be minimized. There are a number of outcomes:

1. **The algorithm fails**, because the final A and B yield a risk that is still very high. This is often due to *local minima* in the risk function. The number of local minima often depends on the complexity of the model used. We used a simple model (Equation 1 and 2), so the local minima should be few.

2. **The algorithm succeeds** because the risk falls to zero (lowest possible value) or a very small number. In this case, the problem is determining if the final values A and B are applicable outside the training period T. If yes, the model is useful because it can be applied at any time to determine rainfall. If not, the model is said to “overfit”, because it fits the training data very well, but gives wrong results if applied at other times.

1.3 Training & Testing

Complex models are prone to overfit, while simple models have the problem that the risk can't be adequately minimized.

Overfit if often the more serious problem of the two because “underfit” is obvious from training while overfit is not. To mitigate the problem of overfit, the data is split into 2 parts:

- a larger “training” portion, usually 70% – 90% of the data and
- a “test” portion, 30% - 10% of the data.

The “accuracy” of the model is reported only on the test data set.

1.4 Gradient Descent

The Risk Minimization algorithm leaves something out – how to “adjust” A and B on subsequent iterations? There are a number of ways to do this, but a popular and well-studied method is called *gradient descent*.

Gradient descent makes changes to a model's parameters where the gradient of the risk function in parameter space is most negative (meaning a downslope).

In our simple model, the two gradients are: $\frac{\partial r}{\partial A}$ and $\frac{\partial r}{\partial B}$ where r is the risk. Suppose we adjusted the parameters using these gradients,

$$A(i+1)=A(i)-\frac{\partial r}{\partial A}(i)$$

Where i is the iteration, and a similar adjustment equation holds for B. Then, from the formula for risk:

$$\begin{aligned} r_{i+1} &\equiv r(A+\delta A, B+\delta B) \\ &= r(A,B)+\delta A\frac{\partial r}{\partial A}+\delta B\frac{\partial r}{\partial A}+O(\delta A^2)+O(\delta B^2) \\ &\approx r(A,B)-(\frac{\partial r}{\partial A})^2-(\frac{\partial r}{\partial A})^2+... \\ &\leqslant r(A,B)\equiv r_i \end{aligned}$$

So, gradient descent causes the risk to decrease at each iteration, which is our goal. Most modern gradient descent algorithms (AdaGrad, Adam, etc.) employ sophisticated descent methods, but the idea behind them is essentially as we've described above.

1.5 Backpropagation

Equation 2 and the Risk it induces can be modeled as a flow of information through a “network” of operations:

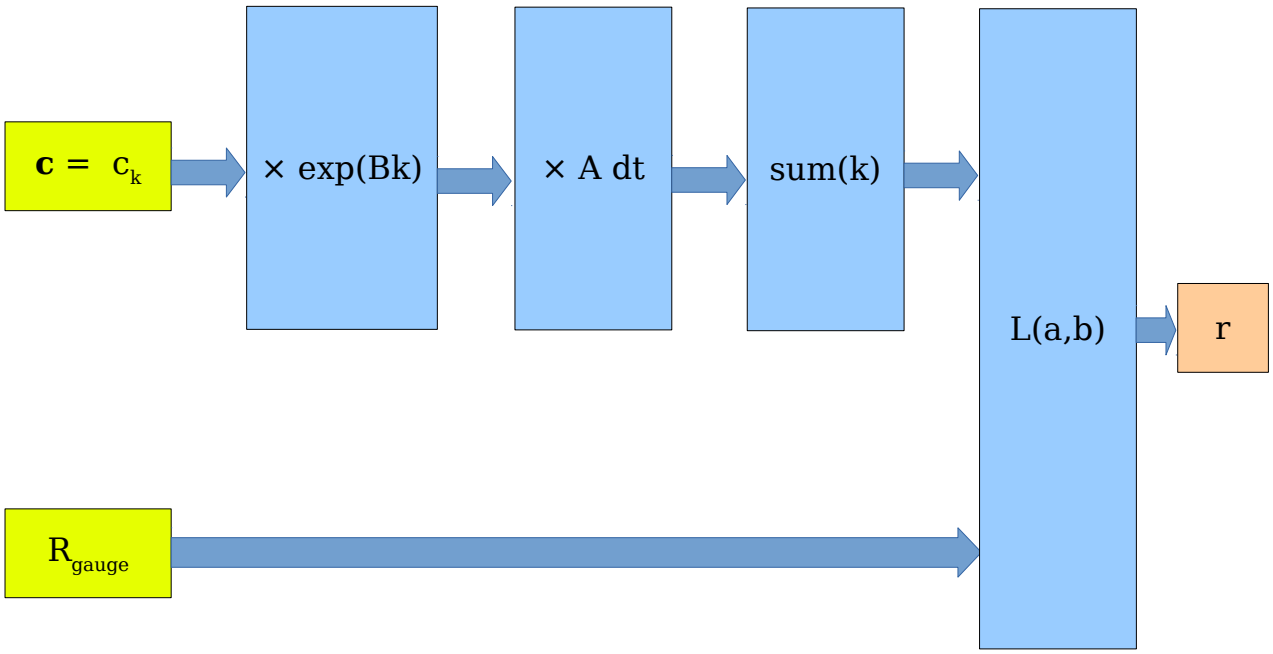


Figure 4: A network representing data flow for the risk algorithm. L(a,b) is the loss function, which we fixed as $L(a,b)=(a-b)^2$ in Listing 1.

In Figure 4, the yellow boxes represent **input** data sources, the orange box on the right represents the **output** risk at each iteration and the blue boxes the sequence of operations that transform input to output.

The *backpropagation algorithm* is a simple way for us to calculate derivatives of the output (risk) with respect to any system parameter (in this case, just A and B).

In the general case, we denote:

- The system's tunable parameters by W (also called the system's *weights*). In our simple example, this is just A and B.
- The input into the *tunable* part of the network is X. In our example, this is **c**, the bin counts.

The key here is to calculate the term $\frac{\partial r}{\partial W}$ for our network. Now,

$$\frac{\partial r}{\partial W}=\frac{\partial E_D[\![L]\!]}{\partial W}=E_D[\![\frac{\partial L}{\partial W}]\!]$$ – Equation 3

Where $E_D[\![L]\!]$ is just the expected value for L over time, which is implied in step 4 of the risk minimization algorithm.

The term $\frac{\partial L}{\partial W}$ is really shorthand for $\frac{\partial L}{\partial W_j^\alpha}$, where W_j^α is the *j*-th parameter for associated with operation s_α. Consider a path $\{s_\alpha, s_{\alpha+1}...s_N\}$ that starts with s_α, and ends with s_N, a operation that outputs the prediction. *L* can therefore be written as:

$$L=L(X_N)=L(X_N(W^N;X_{N-1}))$$

Where X_N is the output of operation s_N and W_N its parameters. Using the chain rule once, we have:

$$\frac{\partial L}{\partial W_j^\alpha}=\frac{\partial L(X_N)}{\partial W_j^\alpha}=\frac{\partial L}{\partial X_N}\frac{\partial X_N}{\partial W_j^\alpha}$$ – Equation 4

The term $\epsilon=\frac{\partial L}{\partial X_N}$ depends only on the form of the loss function. Its value should vary according to the model's performance over the entire training set. Also, since W^α and W^N are independent variables, we have:

$$\frac{\partial X_N}{\partial W_j^\alpha}=\frac{\partial X_N(W^N;X_{N-1})}{\partial W_j^\alpha}=\frac{\partial X_N}{\partial X_{N-1}}\frac{\partial X_{N-1}}{\partial W_j^\alpha}$$

The term $\delta_N=\frac{\partial X_N}{\partial X_{N-1}}$ measures how much the output of operation s_{N-1} affects the output of operation s_N. We can therefore rewrite Equation 4 as:

$$\frac{\partial L}{\partial W_j^\alpha}=\epsilon\frac{\partial X_N}{\partial W_j^\alpha}=\epsilon\,\delta_N\frac{\partial X_{N-1}}{\partial W_j^\alpha}$$

From this, it shouldn't be too hard to convince yourself that:

$$\frac{\partial L}{\partial W_j^\alpha}=\epsilon\,\delta_N\delta_{N-1}...\delta_{\alpha+1}\frac{\partial X_\alpha}{\partial W_j^\alpha}$$ – Equation 5

We call the term $\epsilon_{\alpha+1}=\epsilon\,\delta_N\delta_{N-1}\ldots\delta_{\alpha+1}$ the error received by operation s_α from the path $\{s_{\alpha+1}\ldots s_{N-1}, s_N\}$. You can see from its form that the original error ϵ is **propagated backwards** from s_N into s_α. This is why this algorithm is called "backpropagation".

From this, we can rewrite Equation 5 as:

$$\frac{\partial L}{\partial W_j^\alpha}=\epsilon_{\alpha+1}\frac{\partial X_\alpha}{\partial W_j^\alpha}$$

$$\epsilon_{\alpha+1}=\epsilon\,\delta_N\delta_{N-1}\ldots\delta_{\alpha+1}$$

$$\delta_k=\frac{\partial X_k}{\partial X_{k-1}}$$

$$\epsilon=\frac{\partial L}{\partial X_N}$$

These four equations define the Backpropagation (BP) algorithm. Figure 5 below depicts this flow of error terms:

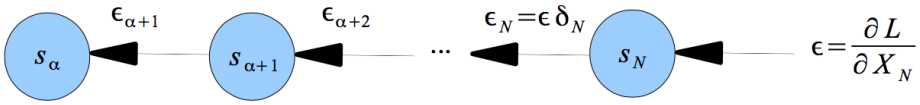


Figure 5 – Error flow in the BP algorithm.

In the event that an operation is the root node of more than one path, then the chain rule means that you need to sum the errors from each path:

$$\epsilon_{\alpha+1}=\sum_{1..\,p}\epsilon_{\alpha+1}^{(p)}$$

Quiz 1 Show this is true.

1.6 Batch, Stochastic and Mini-batch Learning

In *batch learning* or *batch updating* , the weights are only updated once the risk has been calculated for the entire training batch.

The other extreme case is “stochastic” learning” where the weights are updated after receiving each input. Batch learning generally provides the best convergence, and we recommend you start with this. Stochastic learning is generally very “noisy” with sharp peaks in the training loss over time. However, it is said this provides better escape from local minima of gradient descent and much faster convergence.

An alternative form of update between these two extremes is called “mini-batch” learning, where the weight updates are performed after receiving a “batch” (ie, more than 1 but less than the full dataset).

To recover the BP equations involving Risk, we simply have to take the expected value of these equations over the training set *D*:

$$\begin{aligned} \frac{\partial r}{\partial W_j^\alpha} &= E_D\left[\!\left[\epsilon_{\alpha+1} \frac{\partial X_\alpha}{\partial W_j^\alpha} \right]\!\right] \\ \epsilon_{\alpha+1} &= \epsilon \delta_N \delta_{N-1} \ldots \delta_{\alpha+1} \\ \delta_k &= \frac{\partial X_k}{\partial X_{k-1}} \\ \epsilon &= \frac{\partial L}{\partial X_N} \end{aligned}$$

Equations 6 – The BP Equations for Batch Learning

Quiz 2: Apply the batch learning BP equations to calculate $\frac{\partial r}{\partial A}$ and $\frac{\partial r}{\partial B}$ for Figure 4. Use $L(a,b)=(a-b)^2$.

2.0 The Weekly Radar Rainfall Calibration Challenge

Weather radar provides high spatial resolution measurements over a wide region surrounding the radar installation. The flip side of radars is their high sensitivity to calibration errors that limits the usefulness of the information that can be obtained.

On the other hand there is the rain gauges that are much simpler systems and more accurate than radar in measuring rainfall. However they only provide point readings with spatial interpolation required to obtain rainfall estimates away from the rain gauge location.

You will be tackling the radar calibration problem in this competition, using the rain gauge readings to improve the radar backscatter to rainfall rate conversion model. In addition, you will be identifying the locations of the rain gauges in terms of the radar coordinates. Backscatter data from a weather radar will be provided as well as rain gauge readings gathered from various weather stations in Singapore over the past year.

2.1 Provided datasets

Rain gauge
The rain gauge data is provided in a csv file (gauge.csv)

	A	B	C	D	E	F
1	Year	Week	0	1	2	3
2	2017	1	17.2	15.6	12.6	4.2
3	2017	2	0.2	2.4	2	26.8
4	2017	3	68.8	21.2	54.4	67.2
5	2017	4	64.4	106	81.2	70.6
6	2017	5	2.4	7.4	13.8	11.2
7	2017	6	6.4	79.8	47.4	30.4
8	2017	7	3	12.6	6.2	19.6
9	2017	8	26.4	27.2	14.4	29.6
10	2017	9	40.2	45.2	25.6	31.2
11	2017	10	10.2	33.2	8.8	36.2

Figure 5: Gauge data format

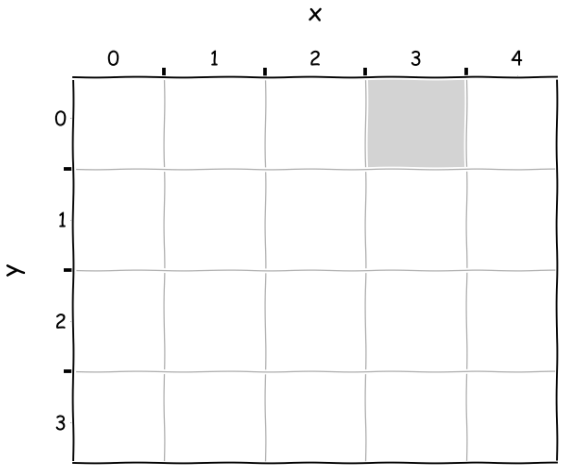
Each row (after the first row) contains the gauge readings for a particular week given in the first two columns. Each column from the third column onwards are rain gauge readings from a particular weather station. Data from 50 rain gauges are provided, accounting for 50 columns in the file. The names of the weather stations are not provided, instead they are simply numbered 0 to 49 in random order.

The gauge readings are accumulated by week from daily values. The gauge readings are not always available every day. Weeks which contain any missing readings are omitted from the weekly data and are marked as a blank field in the csv file.

Radar
The radar backscatter data are in the radar subfolder. Each csv file is named by the year and week of the data it contains, eg 2017-29.csv refers to week 29 of year 2017. The spatial resolution of the radar is a square of side 292m

(Figure 6, left), with a total field size of 480 x 480 squares. The top left corner of the field is at 1.980 deg latitude, 103.338 deg longitude.

288 readings are taken per day which are accumulates to 2016 readings in a week. The readings are not the raw backscatter intensity value but are quantized into 34 bins. Each row in the csv file (from the 3rd column onwards, see Figure 6, right) is thus the bin counts over a week at a particular coordinate given by the first 2 columns of that row.



y	x	0	1	2	3
0	0	1917	0	0	2
0	1	1919	0	0	0
0	2	1918	0	2	2
0	3	1919	0	0	0
0	4	1919	0	0	0
0	5	1918	1	0	0
0	6	1919	0	2	0
0	7	1924	0	0	0
0	8	1918	0	0	2
0	9	1923	0	0	1
0	10	1923	0	0	0
0	11	1928	0	0	0

Figure 6: Radar coordinate system (left), with the square at (3,0) shaded and radar data format (right), with the row corresponding to the (3,0) coordinate highlighted

Note that the radar also has occasional down times leading to missing data. Thus there are some weeks which do not have a csv file. Be careful in how you treat the missing data in both radar and rain gauge data when you perform your analysis!

2.2 Expected output

You are free to use any methods or tools, whether covered in this notes or not to produce the following outcomes:

- Make a determination of where the 50 stations are (their locations) in the radar coordinates
- Improve on the simple model described in the lecture

- With the station location and improved model, calculate the Euclidean distance of the calibrated radar readings from the gauge data

The results of your efforts are to be written up in a report and a slides deck containing the following:

- State the methodology used and substantiation for using it
- Clear description of the model
- Present the model outputs (loss values / graphs)
- Attach code used in an appendix

3.0 Python Guide

We include in this section some useful information on performing common tasks in Python effectively. Note especially the final section in which a class for gradient descent as well as other minimization functions are described.

3.1 Jupyter Notebook

For this challenge, you are encouraged to use Jupyter notebook, a popular Python-based application that allows you to mix descriptive text and equations, python code and their output, all in a single notebook format. This is extremely useful for performing data analysis and interactively presenting your results, whether they are graphical plots, tables or individual numerical values.

Installation
Download the Anaconda distribution from www.anaconda.com/download/

Basics
Start Jupyter notebook by running the command `jupyter notebook` in your terminal or navigate `Start > Anaconda > Jupyter Notebook` for Windows Anaconda users. The Notebook app starts with a dashboard that allows you to navigate the folder structure and create/rename/delete files. See figure below on how to create a new Python notebook.

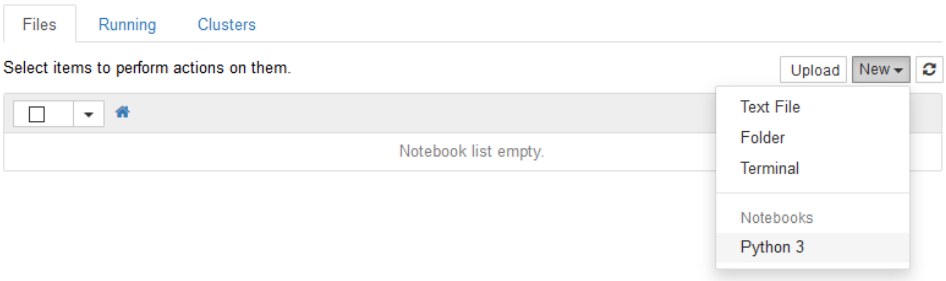
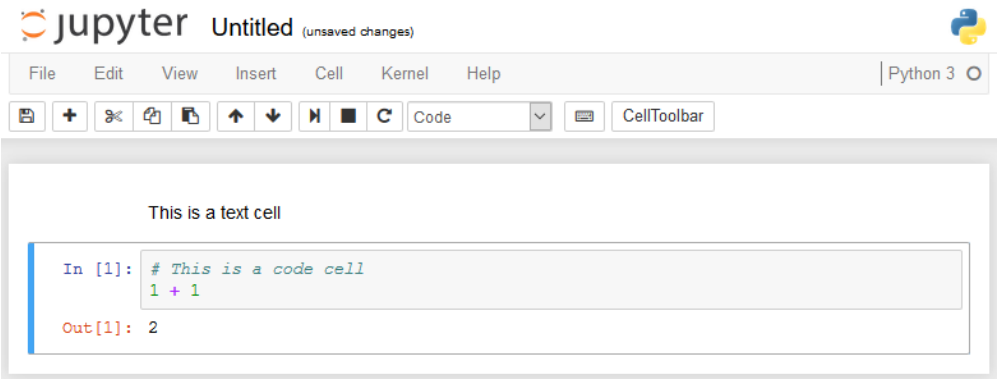


Figure 7: Creating a new Python notebook

Cell types

Jupyter notebooks can contain rich textual information, as well as code blocks. Textual data are stored in 'Markdown' cells while code is stored in 'code' cells.



Markdown cells are so called as they use the Markdown markup language, which has a simple syntax to learn. For example to italicize text, just surround it with *, ie

```
normal text *italicized* normal text
```

For bold text, use double stars, `**` instead of single as for italics. Create new lines with double space at the end of the line. Latex equations can also be added into markdown cells by wrapping the latex code with `$` for inline equations, and `$$` for equations on their own line.

Code cells contain python code. When a cell is run (by pressing shift+enter or ctrl+enter) the code in the cell is executed and any output is displayed right below the code textbox, but is still part of the same cell.

Within a notebooks. the program state, ie the values of variables, are shared among all the cells. So **executing cells in different order may change their outputs** if they rely on variables set in another cell. The state is also not stored in the notebook files. This means when you first open a notebook, it has an empty state until you manually start executing code cells.

Common notebook operations

Create new cells	Insert > Insert Cell Above / Below, or Second button on the toolbar
Cut/Copy/Paste/Delete cell	All these operations have their own buttons on the toolbar and can also be accessed from the Edit menu
Change cell type	Choose the desired cell type from the drop menu in the toolbar
Select multiple cells	Shift+click on the cells *outside the textbox*
Execute cell	Ctrl+enter
Execute cell and select the next cell	Shift+enter
Save notebook	First button on the toolbar
Exit notebook	File > Close and Halt

3.2 Math operations

The numpy library contains many useful functions for scientific computing. The basic data type is the n-dimensional array (ndarray) that can store multi-dimensional data. Operations like +, – , *, / between two ndarrays are applied element-wise.

Calculating correlations
The scipy library has a function to calculate the Pearson correlation coefficient to correlate two variables. The syntax is

```
import scipy
scipy.stats.pearsonr(x,y)
```

where x and y are the two lists or arrays to be correlated. The function returns a list of two values – the correlation coefficient and a 2-tailed p-value.

Plotting graphs

Graphs are plotted with the matplotlib library. You need to prepare an array of x-coordinates and y-coordinates to create a plot. When using Jupyter notebook, you also need to include the line `%matplotlib inline` for the plot to be displayed in the notebook. Sample code for creating a plot of one period of a sine wave is given below

```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

x = np.linspace(0, 2*np.pi, 100)
y = np.sin(x)
plt.plot(x, y);
```

Listing 2: Plotting a sine wave

- Set your figure size using `figsize` option in `plt.figure`, ie

```
plt.figure(figsize=(10,6))
```

- Axis limits are set using the following functions

```
plt.xlim(min_x, max_x)
plt.ylim(min_y, max_y)
```

  where the two parameters are the desired axis limits.

- Axis labels: set them with

```
plt.xlabel('LABEL', fontsize='SIZE')
plt.ylabel('LABEL', fontsize='SIZE')
```

  Here "LABEL" is the text string to set as the axis labels and SIZE can be any of the following: `xx-small, x-small, small, medium, large, x-large, xx-large`

- Saving figures: save your plots with the savefig function.

```
plt.savefig('c:/users/foo.png', bbox_inches='tight')
```

  The first argument here is the path to the file that will be created. The

second option (bbox_inches) is to reduce the whitespace around the plot.

Note that forward slashes are used in the path, which is unlike the normal convention in Windows which uses backslashes to separate folders.

3.3 Table operations

Working with tables is easily done with the pandas library. Remember to import the library before use

```
import pandas as pd
```

The relevant data structures are the Dataframe and Series, which hold 2D data (tables) and 1D data (lists) respectively.

Reading and writing CSV files
To load a CSV file, use the command

```
df = pd.read_csv('c:/path/gauge.csv')
```

Note again the forward slashes in the path name. This loads the CSV data into a Dataframe and stores it as a variable called `df`.

To load all files from a folder of CSV files at once, use the `listdir` command from the `os` library together with `read_csv`

```
import os
files = os.listdir(directory_path)
loaded = [pd.read_csv(i) for i in files]
```

This loops over all the files in the directory and results in a list of Dataframes stored as the `loaded` variable.

Write Dataframes to CSV files with the `to_csv` function

```
df.to_csv('c:/path/new.csv')
```

Slicing Dataframes

There are two methods to selecting individual or groups of cells, `loc` and `iloc`. `iloc` refers to cells by their numerical coordinates, while `loc` uses the column and row labels. These methods apply as well to Series – just don't include the second dimension in the commands.

- `df.iloc[1,2]` selects the cell in the second row (row 1), third column (column 2)
- `df.loc[2017,'3']` selects the cell in the row with the numerical label 2017 and column with string label '3'

Groups of cells can be selected by slicing in one or both dimensions within specified ranges. The slices are given as Series or DataFrames as appropriate.

- `df.iloc[3:6, 1:]` selects the fourth to sixth rows, second column onwards
- `df.loc[2017,'1':'3']` selects the cells in row 2017 from column '1' to column '3' inclusive

Observe in the example above the slice ranges specified with the colon : notation. Note further that the start or end range limit may be omitted to select to the end/beginning of the row/column. If both are omitted then the entire row/column is selected.

Negative index may also be used in the case of `iloc` to start counting from the end

- `df.iloc[:, -3:-1]` selects the second- and third-last columns of all rows in the Dataframe

3.4 Gradient descent and backpropagation

A simple implementation of the gradient descent and backpropagation algorithm is provided in the grad-descent.ipynb notebook. Two classes are defined, Operation and Path. These classes are not built into Python, so to use them you need to copy the code in the first cell of the provided notebook and run it first.

Operation

This class represents a node in the multi-stage network for backpropagation. It handles the forward and backward passes, storing the output and errors in internal variables, as well as update its weights as part of the backward pass.

To declare an operation without weights,

```
Operation(forward, backward)
```

where forward and backward are functions that will be executed in the forward and backward passes respectively.

For operations with a weight, the learning rate, initial value for the weight and the weight error function have to be given:

```
Operation(forward, backward, learning_rate, [w_initial],  [dx/dw])
```

Note the square brackets around the final two parameters. In this implementation, each operation can only have one scalar weight.

Path

The path links all the operations together, handling the passing of outputs and errors between operations, and also adds on a euclidean loss operation at the end. To declare a path, you need the list of operations in order, as well as the inputs to feed to the first operation of the path and the desired output (labels)

```
Path([list of operations],input to first operation, label)
```

The forward and backward function of the path can be called to run the respective functions for all operations in the path. Use them in a loop to perform multiple iterations of gradient descent

```
for i in np.arange(50):
    p.forward()
    p.backward()
```

The forward function returns a list containing the output of the final operation and the loss for that iteration. These can be saved in a list and plotted (refer to the provided Jupyter notebook file).

The optimized weights can be individually obtained from their respective Operation class instances, eg

```
op3.weights
```

where `op3` is a previously declared operation.

Other optimization functions

The Scipy library provides several optimization functions which you might find useful. For example, the `scipy.optimize.minimize` function allows you to choose between the Nelder-Mead, Powell, CG, BFGS, Newton-CG and 8 other optimization methods.