Week 1

Introduction

1.1 Overall Objectives

In these laboratory sessions you will explore some of the computational intelligence techniques that have been covered in the first part of this course. These laboratories are self-explanatory and should serve as an introduction to the tools that will be required in your coursework. In particular, this laboratory series will enable you to:

— Write, run, and debug Python using Visual Studio
— Build a simple perception model of a neuron
— Assemble an artificial neural network using the perception model
— Implement back-propagation training for your artificial neural network
— Implement PCA and a *k*-nearest neighbour classification system

> **Important Message**
>
> This laboratory series builds on the knowledge developed within the lectures; you should make sure you are comfortable with the lecture content before attempting these laboratories. It does not assume any prior knowledge of Python and has a gentle introduction.

2.1 Week One Objectives

In this session you will learn how to create, run, and debug a Python program from within Microsoft Visual Studio. Your objectives are:

1. Create a Python project using either Visual Studio Code or Visual Studio
2. Run a basic Hello World example and become familiar with the editor
3. Write and test some simple programs that explore the key language concepts of Python
4. Learn how to use the debugger as a tool for software development

At the end of this laboratory session you should feel comfortable with creating, editing, debugging and running your own Python program. Note that this course has students from many different backgrounds and so while some will already be familiar with Python there are others who will never have used it before, and so this lab starts with the very basics.

2.2 Introduction

Personal Computers

Python and Visual Studio Code are both free software, so you can install them on your own computers for free. Python is available for pretty much every operating system and platform, but there are two caveats to consider:

1. Python underwent a major version shift between Python 2.7 and Python 3.x, the differences were big enough that Python 3.x is backwards incompatible with Python 2.7. In this course we will be using **Python 3.8** and this is the version that you should install on your machine.
2. Python itself is just the core language, and throughout this course we will make use of many different libraries that extend the core functionality. You can install different libraries individually, but the easiest way is to install a packaged distribution of Python and the major libraries, I recommend using Anaconda.

Thus, the best way to install Python on your home computer is to install Anaconda for Python 3.8 from: https://www.anaconda.com/download/

You are free to use whatever IDE (Integrated Development Environment) you wish. The University computers use Visual Studio and I recommend the free version called Visual Studio Code, available from: https://code.visualstudio.com/

Visual Studio Code and Anaconda support Windows, MacOS, and Linux. Basic instructions are provided via a short video on Moodle for running and debugging code, but the staff and lab demonstrators have experience of other platforms and will be happy to help. You can access the video introduction from:
https://uniofbath.cloud.panopto.eu/Panopto/Pages/Viewer.aspx?id=6bcf89a9-6f64-4890-b7fc-ac4300e159a9

University Computers

Python and Visual Studio are also available on the University computers, including in the library and via Remote Desktop to UniDesk. More information on connecting to UniDesk can be found from: https://www.bath.ac.uk/guides/work-remotely-with-uniapps-and-unidesk/

2.3 Some Examples

In order to introduce you to some of the key Python concepts I have provided some short code examples that show you the core aspects of the language. These are very simple, you do not need to try each one out, but you may find it useful to experiment with a few that you find more difficult. These examples can be used as a "cheat sheet" for more complicated programs. If you want to try them, you may either create new files for each example or you could add them sequentially to your existing .py file. Remember that you can print a variable to the console by using the *print* command, if you want to see what the contents might be.

Most of these examples are taken from the very good Python tutorial, you can look there for more detailed notes: https://docs.python.org/3/tutorial/

Variables and Mathematics

The first example to consider is how we store and manipulate variables (numbers, strings, etc). Python is very clever, and so unlike languages such as C it can work out what the type of your variable is from the contents – there is no need to explicitly type your variables. The code listing below shows some examples of creating and modifying some simple numerical variables.

Numerical

```
1  # This is a comment!
2
3  # Python knows that pi is a float
4  pi = 3.141
5
6  # and that freq is an integer
7  freq = 20000
8
9  # So nextFreq will also be an integer
10 nextFreq = freq + 1
11
12 # and radians/s will be a float, because we used a float in the calculation
13 rads = 2 * pi * nextFreq
14
15 # Classic division always returns a float
16 subRads = rads / 2.4
17
18 # Floor division returns the integer part (discards the fraction)
19 # Modulo division returns the fractional part (discards the integer part)
20 subRadsInt = rads // 2.4
21 remainder = rads % 2.4
22
23 # Powers can be calculated using **, here we square the frequency
24 f2 = freq ** 2
```

Code 2.1: numerical.py

Strings

```
# Strings can use either single or double quotes
firstName = "Bob"
lastName = 'Widlar'

# but be careful if you need to "contain" a quote mark, use \ toescape # lastName = Mc'Gee
lastName = 'Mc\'Gee'

# String can be concatenated using +
fullName = firstName + " " + lastName

# Strings can be indexed, with the first character having index [0].
# There is no character type, a character is simply a string of length 1
firstLetter = fullName[0]

# You can also index from the right, so [-1] is the last character (-0 =0)
lastLetter = fullName[-1]

# If you just want a slice, you can give a range # Starting at 0 and ending at 2
firstName = fullName[0:3]

# You can't change modify a string, but you can over-write it or make a new one # You can use len() to check the size..
nameLength = len(fullName)
```

Code 2.2: strings.py

Lists

```
# Lists are a way of storing lots of elements
squares = [1, 4, 9, 16, 25]

# Just like strings they can be indexed (from 0) and sliced
one = squares[0]
end = squares[-1]

# Note that the slice operation returns a new list containing theelements
squaresSlice = squares[0:3]

# Lists can be concatenated
squares = squares + [36, 49, 64, 81, 100]

# Unlike strings, lists are mutable. You can change the individual elements
squares[0] = 1 ** 2

# Len() also works on lists...
squaresLength = len(squares)
```

Code 2.4: lists.py

Loops and Flow Control

Loops and flow control are the next important area. You will have noticed by now that unlike C, Python does not require a semi-colon at the end of each line of code, or a set of curly brackets around code blocks. When want to group together bits of code, for example inside a loop, then we do so by indenting the code use a TAB space.

Loops

```
# Loops are a great way of repeating sections of code # Can you work out what this does?
a = 0
b = 1

# Importantly, the contents of the loop is enclosed using indents (single tab)
# Not {} like you get in C, loops can be nested, you just keep indenting
while b < 10:
    print(b)
    a = b
    b = b + a

# We can also use for loops
# We need to generate a list of items to loop through
words = ['cat', 'window', 'defenestrate']
for word in words:
    print(word)

# If we just want to loop a set number of times, we can make a range of integers # An easy way to this is using range()
# range(10) produces [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
for i in range(10):
    print(i)

# We can perform a conditional check using the if statement
# the else and the elif (else if) are optional, note the indentation to group the code
if pi < 3:
    pi = 3.141
    print("Who ate the pie?")
elif pi > 4:
    print("Too much pie..")
else:
    pi = 0;
```

Code 2.5: loops.py

User Input

User Input

```
# We have already seen how to use print() to display data
# We can use input to get some data from the console
response = input("Is Python amazing? ")
print("You said: " + response)
# If we want to convert from string to int, or vice versa
limit = int(input("Enter a limit: "))
if limit > 3:
    # We could also have used "Your limit of", limit, "was too big"
    print("Your limit of " + str(limit) + " was too big")
```

Code 2.6: input.py

2.4 Debugging

One of the great features of Visual Studio Code is the excellent debugger, it will let us run our code one line at a time and inspect the value of all of our variables. In order to run the code line by line we need to insert a *breakpoint*, this tells Python to run from the start of our code until this line and stop there. You can insert a breakpoint by double clicking on the bar at the left-hand side of the editor window. Once in place a red circle will indicate the breakpoint, and when you run your code it will now stop at this point.

If you now hover the cursor over a variable it will show you the current value of that variable, you can also use the watch window (select Locals) to see the current value of all variables. Using the run options tool bar, you can either run to the next breakpoint, or single step the code one line at atime and watch as the variables change state.

2.5 Exercises

Spend some time familiarising yourself with the Visual Studio Code environment and the debugger, it will prove a valuable asset in the rest of the course. If you want some more challenging exercises, see if you can produce software to perform the following functions:

1. Given a number entered by the user, compute and display the factorial.
2. Store a secret password, ask the user to enter a password and compare the two (hint: strings are compared just like integers in C or MATLAB).
3. Display the maximum value that can be stored in a numerical variable.