3. Testing your installation

Bookmark this page

Testing your installation

Download project0.tar.gz and untar it in to a working directory. To deal with tar.gz files on windows, you can use 7-zip.

The `project0` folder contains two pyhon files.

- **main.py** contains the various functions you will to complete in the next sections of the project
- **test.py** is a script which runs tests
- **debug.py** contains the code for the final problem of this project

Tip: Throughout the whole online grading system, you can assume the NumPy python library is already imported as np.

This project will unfold both on MITx and on your local machine. You are welcome to implement functions locally and then copy+paste your code into the MITx code boxes to fully check correctness and receive your grade for individual function implementations. Alternatively, you can also implement the functions online first and after finishing, copy+paste the solution to your local **main.py** file. Be wary of the number of attempts you have for each problem, especially if you choose the second development flow.

How to Test Locally: In your terminal, navigate to the directory where your project files reside. Execute the command `python test.py` to run all the available tests.

Randomization

1.0/1 point (graded)

Write a function called `randomization` that takes as input a positive integer `n`, and returns `A`, a random n x 1 Numpy array.

Available Functions: You have access to the NumPy python library as np

Grader note:: If the grader appears unresponsive and displays “Processing", it means (most likely) it has crashed. Please resubmit your answers, and leave a message in the forum and we will work on fixing it as soon as possible.

```
def randomization(n):
    """
    Arg:
      n - an integer
    Returns:
      A - a randomly-generated nx1 Numpy array.
    """
    #Your code here
    A = np.random.random([n, 1])
    return A
    raise NotImplementedError


```

Press ESC then TAB or click outside of the code editor to exit

Correct

Operations

1.0/1 point (graded)

Write a function called `operations` that takes as input two positive integers `h` and `w`, makes two random matrices `A` and `B`, of size h x w, and returns `A`, `B`, and `s`, the sum of `A` and `B`.

Available Functions: You have access to the NumPy python library as np

Grader note:: If the grader appears unresponsive and displays "Processing", it means (most likely) it has crashed. Please resubmit your answers, and leave a message in the forum and we will work on fixing it as soon as possible.

```
6     Arg:
7       h - an integer describing the height of A and B
8       w - an integer describing the width of A and B
9     Returns (in this order):
10      A - a randomly-generated h x w Numpy array.
11      B - a randomly-generated h x w Numpy array.
12      s - the sum of A and B.
13    """
14    #Your code here
15    import numpy as np
16    A = np.random.random([h,w])
17    B = np.random.random([h,w])
18    s = A + B
19    return A, B, s
20    raise NotImplementedError
21
```

Press ESC then TAB or click outside of the code editor to exit

Correct

Norm

1.0/1 point (graded)

Write a function called `norm` that takes as input two Numpy column arrays `A` and `B`, adds them, and returns `s`, the L2 norm of their sum.

Available Functions: You have access to the NumPy python library as np

Grader note:: If the grader appears unresponsive and displays “Processing", it means (most likely) it has crashed. Please resubmit your answers, and leave a message in the forum and we will work on fixing it as soon as possible.

```
2
3     Takes two Numpy column arrays, A and B, and returns the L2 norm of their
4     sum.
5
6     Arg:
7       A - a Numpy array
8       B - a Numpy array
9     Returns:
10      s - the L2 norm of A+B.
11    """
12    #Your code here
13    import numpy as np
14    s  = np.linalg.norm(A + B)
15    return s
16    raise NotImplementedError
17
```

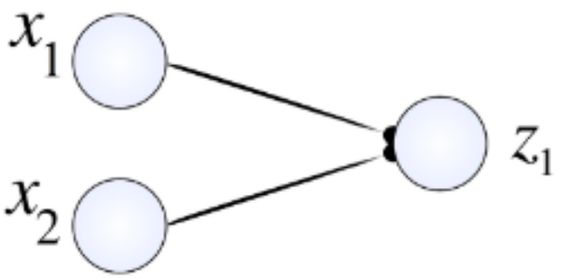
Press ESC then TAB or click outside of the code editor to exit

Correct

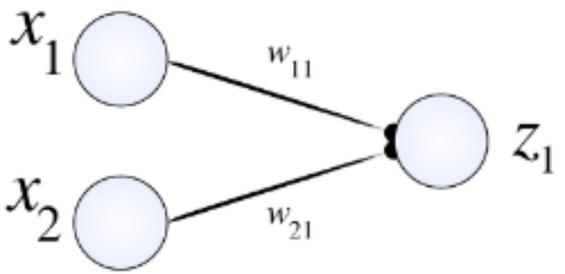
Project0 due Sep 22, 2020 19:59 EDT **Completed**

As introduced in the previous section, a neural network is a powerful tool often utilized in machine learning. Because neural networks are, fundamentally, very mathematical, we'll use them to motivate Numpy!

We review the simplest neural network here:



The output of the neural network, z_1, is dependent on the inputs x_1 and x_2. The importance of each of the inputs is given by values called *weights*. There is one weight from each input to each output. We show this here:



The inputs are given by x, and the outputs are given by z_1. Here, w_{11} is the weight of input 1 on output 1 (our only output in this case), and w_{21} is the weight of input 2 on output 1. In general, w_{ij} represents the weight of input i on output j.

Your function should take two arguments: `inputs` and `weights`, two NumPy arrays of shape $(2, 1)$ and should return a NumPy array of shape $(1, 1)$, the output of the neural network. Do not forget the $\tanh$ activation.

Grader note:: If the grader appears unresponsive and displays "Processing", it means (most likely) it has crashed. Please resubmit your answers, and leave a message in the forum and we will work on fixing it as soon as possible.

```
def neural_network(inputs, weights):
    """
     Takes an input vector and runs it through a 1-layer neural network
     with a given weight matrix and returns the output.

     Arg:
       inputs - 2 x 1 NumPy array
       weights - 2 x 1 NumPy array
     Returns (in this order):
       out - a 1 x 1 NumPy array, representing the output of the neural network
    """
    #Your code here
    z = np.tanh(inputs.T @ weights)
    return z
    raise NotImplementedError

```

Press ESC then TAB or click outside of the code editor to exit

Correct

Test results

6. Vectorize function

Bookmark this page

Project0 due Sep 22, 2020 19:59 EDT *Completed*

In this exercise, you will learn how to vectorize a function that can only deal with scalar inputs without using a `for` loop.

Scalar function

1.0/1 point (graded)

Let's start with writing a scalar function `scalar_function`, which will apply the following operation with input `x` and `y`.

$$f(x,y) = \begin{cases} x \cdot y, \text{ if } x \leq y \\ x/y, \text{ else.} \end{cases}$$

Note that `x` and `y` are scalars.

Available Functions: You have access to the NumPy python library as np

Grader note:: If the grader appears unresponsive and displays “Processing", it means (most likely) it has crashed. Please resubmit your answers, and leave a message in the forum and we will work on fixing it as soon as possible.

```
def scalar_function(x, y):
    """
    Returns the f(x,y) defined in the problem statement.
    """
    #Your code here
    if ( x <= y ):
        return x*y
    else:
        return x/y
    raise NotImplementedError

```

Press ESC then TAB or click outside of the code editor to exit

Correct

```
def scalar_function(x, y):
    """
    Returns the f(x,y) defined in the problem statement.
    """
    if x <= y:
        return x*y
    else:
        return x/y
```

Test results

Vector function

1.0/1 point (graded)

`scalar_function` can only handle scalar input, we could use the function `np.vectorize()` turn it into a vectorized function. Note that the input argument of `np.vectorize()` should be a scalar function, and the output of `np.vectorize()` is a new function that can handle vector input.

Please write a vector function `vector_function`, which will apply the operation $f(x, y)$ defined above element-wisely with input vectors with same dimension `x` and `y`.

Available Functions: You have access to the NumPy python library as np, and the function `scalar_function` in the previous exercise.

Grader note:: If the grader appears unresponsive and displays “Processing", it means (most likely) it has crashed. Please resubmit your answers, and leave a message in the forum and we will work on fixing it as soon as possible.

```
def vector_function(x, y):
    """
    Make sure vector_function can deal with vector input x,y
    """
    #Your code here
    vfunc = np.vectorize(scalar_function)
    return vfunc(x,y)
    raise NotImplementedError

```

7. Introduction to ML packages

Bookmark this page

In the resources tab of the course, we have provided you with two notebooks.

Introduction to ML packages (part 1) Github - Notebook viewer
Introduction to ML packages (part 2) Github - Notebook viewer

They cover some of the most useful ML packages and constitute a good reference point to refer to as you progress through the course.

We do not expect you to complete all sections in these notebooks immediately. For now, go through the first three sections in the first notebook on **Jupyter**, **Numpy**, and **Matplotlib**. Then after Unit 1 *Linear Classifiers*, come back to the section on **Scikit learn**, and while you work on Unit 3 *Neural Nets*, refer to the second notebook, which gives an introduction to **Pytorch**.

We will not be using **Pandas** in this course, but it is a useful tool. Feel free to look at the section on Pandas at any time.

By the end of the course, our hope is that you are able to recreate the content of these notebooks by yourself ...and more!

8. Debugging exercise

Bookmark this page

Project0 due Sep 22, 2020 19:59 EDT *Completed*

In machine learning, there a lot of reasons why a model would fail to perform well. The model could be poorly designed, the data could be too noisy, it could just be a poor initialization, or a poor choice of hyperparameters, etc. Therefore, it is vital to at least exclude engineering bugs using a debugger.

Fortunately, Python comes with a fully functional interactive debugger called pdb.

Before you tackle the last problem of this project, we recommend you take some time to learn how to use, for example with this tutorial.

Debugging exercise

1.0/1 point (graded)

In this problem, you are given a buggy piece of code and are asked to debug it.

The goal of this exercise is for you to set up a working debugging system for yourselves. (See the next page for an example.) Feel free to use other debuggers as you wish. But note that any extra print statement in submitted code will be graded as incorrect in the online code graders.

The function `get_sum_metrics` takes two arguments: a `prediction` and a list of `metrics` to apply to the prediction (say, for instance, the accuracy or the precision). Note that each metric is a function, not a number. The function should compute

each of the metrics for the prediction and sum them. It should also add to this sum three default metrics, in this case, adding 0, 1 or 2 to the prediction.

Reminder: You should fix this function locally first, and run `python debug.py` in your `project0` directory to make sure it behaves as expected on the simple test cases provided

Grader Note: : If the grader appears unresponsive and displays “Processing", it means (most likely) it has crashed. Please resubmit your answers, and leave a message in the forum and we will work on fixing it as soon as possible.

```
def get_sum_metrics(predictions, metrics=None):
    if metrics is None:
        metrics = []

    for i in range(3):
        metrics.append(lambda x, i=i: x + i)

    sum_metrics = 0
    for metric in metrics:
        sum_metrics += metric(predictions)

    return sum_metrics

```

Press ESC then TAB or click outside of the code editor to exit

Correct

Test results