Lab 1 Exercises for COMP 432 Machine Learning

This is a Jupyter Notebook.

When you opened it, the Jupyter server started a new Python interpreter to run the code you type in the cells.

- **To edit a code cell**, click inside its text box. A green border should appear.
- **To run the selected cell**, hit Ctrl+Enter.

Try typing `3+4` into the cell below and run it with Ctrl+Enter. You should see `7` appear in the output.

In [149…

```
3+4
```

Out[149…

```
7
```

This lab notebook has three types of cells:

1. **Instruction cells** explain what you're supposed to do; don't bother trying to edit them, even though you can select them.
2. **Code cells for answers** are where you should write your code answers, after reading the instructions.
3. **Code cells for checking answers** come with code and can be run to help you check your answer, if applicable.

Before attempting the exercises, you should have already read the Lab1 overview document.

---

1. Python language exercises

Exercises 1.1–1.4 help to asses your understanding of the Python programming language and how it works. They assume intermediate-level Python expertise, so beginners will find them very challenging, but that is OK — keep learning until they make sense! Use the Python language review and ask TAs for help.

Exercise 1.1 – Python variables, objects, and references

The Python language review for this course contains a diagram of how a list object holds references to the other objects that are "in" the list. Go look at the example diagram in the "List" section.

Now consider the diagram below: image It depicts two list objects, an integer object, a float object, and a string object.

Write Python code that creates objects in memory as depicted in the diagram, including the references shown. Write your code in the cell below. Aim for 3 lines of code, and try not to create any extra (temporary) objects not shown.

In [150…

```
x = [1, 2.0, "+3"]
y = [x[2]]
y.append(y)
```

Check your answer by running the code cell below.

In [151…

```
assert 'x' in globals(), "You didn't define variable 'x'"
assert 'y' in globals(), "You didn't define variable 'y'"
assert isinstance(x, list), "Variable 'x' should refer to a list object"
assert isinstance(y, list), "Variable 'y' should refer to a list object"
assert len(x) == 3, "The list object referred to by 'x' should have length 3"
assert len(y) == 2, "The list object referred to by 'y' should have length 2"
assert isinstance(x[0], int)   and x[0] == 1,   "Slot x[0] should refer to an in
assert isinstance(x[1], float) and x[1] == 2.0, "Slot x[1] should refer to a flo
assert isinstance(x[2], str)   and x[2] == '+3', "Slot x[2] should refer to a st
assert y[0] is x[2], "Slot y[0] should refer to the same object as slot x[2]"
assert y[1] is y,    "Slot y[1] should refer to the same list object that variab
print('Correct!')
```

```
Correct!
```

Exercise 1.2 – Filtering a Python list

Suppose you are given a sequence *x* containing numbers. You are asked to write a function that returns a new sequence containing only those items that were *strictly above* a certain threshold. In other words, your function should 'filter' the sequence according to a threshold.

Write three versions of the function:

1. The first version should use a standard Python for-loop and return a list. Aim for 5 lines of code inside the function.
2. The second version should use list comprehension and return a list. Aim for 1 line.
3. The third version should use the built-in *filter* function to return an iterator. Aim for 1 or 2 lines.

Enter all three answers in the code cell below.

In [152…

```
def filter1(x, threshold):
    newList = []
    for num in x:
        if num > threshold:
            newList.append(num)
    return newList

def filter2(x, threshold):
    return [num for num in x if num > threshold]
```

```
def filter3(x, threshold):
    return filter(lambda num: (num > threshold), x)
```

Check your answer by running the code cell below.

In [153…

```
x = [1, 9, 2, 8, 3, 7, 4, 6, 5]
for threshold in range(10):
    y = filter1(x, threshold)
    assert isinstance(y, list), "filter1 was supposed to return a list object"
    assert len(y) == len(x) - threshold, "filter1 returned wrong number of items
    assert all([yi > threshold for yi in y])
for threshold in range(10):
    y = filter2(x, threshold)
    assert isinstance(y, list), "filter2 was supposed to return a list object"
    assert len(y) == len(x) - threshold, "filter2 returned wrong number of items
    assert all([yi > threshold for yi in y])
for threshold in range(10):
    y = filter3(x, threshold)
    assert isinstance(y, filter), "filter3 was supposed to return a filter objec
    y_greater = [yi > threshold for yi in y]
    assert len(y_greater) == len(x) - threshold, "filter3's sequence had the wro
    assert all(y_greater)
print("Correct!")
```

```
Correct!
```

Exercise 1.3 – Storing a Python dictionary to a file

Suppose you ran a machine learning experiment and found that the following configuration worked best:

```
learning_rate = 0.01
training_steps = 350
weight_decay = 0.05
```

(Don't worry about what these variable names actually mean, you will by the end of the course.)

Write code to save these settings in a file. You should put the above values in a Python dictionary object, then save the object to a file called `exercise13.pkl` using the `pickle` module. See the "Serialization" section of the Python language review for an example.

(Note that in practice it's better and more secure to store human-readable configurations as a JSON file, rather than a binary format like pickle, but start by learning pickle!)

In [154…

```
# Your code here
import pickle

results = {
    'learning_rate': 0.01,
    'training_steps': 350,
    'weight_decay': 0.05}
```

```
with open('exercise13.pkl', 'wb') as f:
    pickle.dump(results, f)
```

Check your answer by running the code cell below.

In [155…

```
assert 'pickle' in globals(), "You forgot to import the pickle module!"
import os
assert os.path.exists('exercise13.pkl'), "File 'exercise13.pkl' doesn't seem to
with open('exercise13.pkl', 'rb') as f:
    config = pickle.load(f)
assert isinstance(config, dict), "Expected a single dictionary object in the fil
assert len(config) == 3, "Expected 3 keys in the dictionary!"
for key, value in zip(('learning_rate', 'training_steps', 'weight_decay'), (0.01
    assert key in config, "Expected '%s' to be a key in the dictionary" % key
    assert config[key] == value, "Expected value for '%s' to be %s" % (key, valu
print("Correct!")
```

```
Correct!
```

Exercise 1.4 – Python lambda functions and closures

Much of machine learning is based on computing derivatives. The derivative of $f(x)$ can be approximated by *finite differences*. Here is the *central difference:*

$$f'(x) = \lim_{\epsilon \to 0} \frac{f(x+\epsilon) - f(x-\epsilon)}{2\epsilon}$$

For sufficiently small ϵ the above fraction is a good approximation to $f'(x)$. (You should already know this stuff from introductory calculus.) In the computer, we'll use a small number like $\epsilon = 10^{-5}$.

Write a Python function that accepts a function f and returns a function that approximates f' at any given x. Use Python's *lambda* feature to define the new function that you return. Aim for 1 line of code inside the function.

In [156…

```
def approx_derivative_of(f, epsilon=1e-5):
    """Returns a new function that approximates the derivative of f at any point

    Arguments:
    f       -- the function f(x) whose derivative we wish to approximate
    epsilon -- the step size to use for central differencing (default 1e-5)
    """
    # Your code here
    return (lambda x: (f(x + epsilon) - f(x - epsilon)) / (2 * epsilon))
```

To play around with your answer, you can try running the code cell below for different values of *x* or even for different functions.

In [157…

```
def f(x):                # A function to compute x^3
    return x**3

def df_exact(x):         # A function to compute the exact derivative of f, which
```

```
    return 3 * x**2

df_approx = approx_derivative_of(f)  # A function to compute approximate derivat

x = 4.0
print(x, f(x), df_exact(x), df_approx(x))
```

```
4.0 64.0 48.0 47.99999999960391
```

Check your answer by running the code cell below

In [158…

```
def f(x): return x**3
def df_exact(x): return 3 * x**2
df_approx = approx_derivative_of(f, 1e-15)

assert callable(df_approx), "You didn't return a function object!"
tests = [(0., 1.0000000000000003e-30), (1., 3.1641356201816957), (2., 11.9904086
for x, answer in tests:
    exact = df_exact(x)
    approx = df_approx(x)
    assert isinstance(approx, float), "Wrong type! Expected your function to ret
    assert approx == answer, "Wrong value for approximation! Expected %s but got
print("Correct!")
```

```
Correct!
```

---

2. Numpy exercises

First, in the code cell below, write a line of code to import the Numpy package in the standard way, then run the code cell.

In [159…

```
 # Import the numpy module here
import numpy as np
```

If you imported Numpy correctly, you should be able to run the code cell below without error.

In [160…

```
assert 'numpy' not in globals(), "You didn't import numpy the standard way. Do K
assert 'np' in globals(), "You didn't import numpy the standard way. Do Kernel->
print("Ready!")
```

```
Ready!
```

Exercise 2.1 – Numpy slicing and data sharing

The Numpy review for this course explains how Numpy arrays are represented in memory, and how multiple `ndarray` objects can share the same data in memory.

Consider the diagram below, depicting three ndarray objects and two chunks of array data:
image

Notice that the strides attribute of the ndarray object referenced by *x*, *y*, and *z* are all different. You should understand why. For example, if you print *y*, the -3 and -7 elements are swapped from how they appear in the figure above:

```
>>> y
array([[-2., -7.],
       [-3., -8.]], dtype=float32)

>>> z
array([[-2., -7.],
       [-3., -8.]], dtype=float32)
```

Write Python code that creates objects in memory as depicted in the diagram. Write your code in the cell below. Do NOT create any Python list objects in your solution, not even temporarily or as arguments to the *np.array* function. *Hint:* You should use the *arange*, slicing, transpose (*T*), negation, assignment, and *copy* operations provided by Numpy.

In [161…

```
# Your code here. Aim for 4-6 lines.
x = np.arange(15, dtype=np.float32).reshape(3, 5)
x[0:2, 2:4] *= -1
y = x[0:2, 2:4].T
z = np.copy(y, order='C')
```

Check your answer by running the cell below.

In [162…

```
assert 'x' in globals(), "You forgot to create a global variable 'x'!"
assert isinstance(x, np.ndarray), "x should be an ndarray object"
assert x.dtype == np.float32, "the dtype of x should be float32"
assert x.shape == (3, 5), "x shape is wrong"
assert x.strides == (20, 4), "x strides are wrong"
assert 'y' in globals(), "You forgot to create a global variable 'y'!"
assert isinstance(y, np.ndarray), "y should be an ndarray object"
assert y.dtype == np.float32, "the dtype of y should be float32"
assert y.shape == (2, 2), "y shape is wrong"
assert y.strides == (4, 20), "y strides are wrong"
assert x.sum() == 65, "the data for x seem wrong"
assert y.sum() == -20, "the data for y seems wrong"
assert 'z' in globals(), "You forgot to create a global variable 'z'!"
assert isinstance(z, np.ndarray), "z should be an ndarray object"
assert z.shape == (2, 2), "z shape is wrong"
assert z.strides == (8, 4), "z strides should be C-order (see np.copy)"
assert np.array_equal(z, y), "the data for z seems wrong"
assert np.may_share_memory(x, y), "y should be a view into x"
assert not np.may_share_memory(y, z), "z should have its own copy of the data"
print("Correct!")
```

```
Correct!
```

Exercise 2.2 – Use Numpy to write vectorized code

Suppose you are given the function below, where *x* is a two-dimensional ndarray of numbers. You can assume *x* is not empty (has at least one item).

In [163…

```
def exercise22_loop(x):
    n, m = x.shape
    v = x[0,0]
    for i in range(n):
        for j in range(m):
            if x[i,j] > v:
                v = x[i,j]

    y = np.empty_like(x)
    for i in range(n):
        for j in range(m):
            y[i,j] = x[i,j] - v

    return y
```

You should know Python and Numpy well enough to figure out what the above code does.

Write a new version of the function that uses Numpy in a way that doesn't require Python for-loops. In other words, your solution should be completely *vectorized*. This is similar to writing good Matlab code. Write your answer in the code cell below. Aim for 1 line of code.

In [164…

```
def exercise22_vectorized(x):
    return x - np.max(x)
```

Check your answer by running the code cell below.

In [165…

```
assert 'exercise22_loop' in globals(), "You forgot to run the code cell with the
assert 'exercise22_vectorized' in globals(), "You forgot to run the code cell wi
for i in range(10):
    x = np.random.randint(100, size=(5, 3))
    y = exercise22_vectorized(x)
    y_correct = exercise22_loop(x)
    assert isinstance(y, np.ndarray), "You didn't return an ndarray object!"
    assert y.shape == x.shape, "You returned an ndarray but of the wrong shape!"
    assert y.dtype == x.dtype, "You returned an ndarray but of the wrong dtype!"
    assert np.array_equal(y, y_correct), "Wrong result!\nx:\n%s\nexpected:\n%s\n
print("Correct!")

import timeit
x = np.random.randint(1000, size=(200, 200))
loop_time = timeit.timeit('exercise22_loop(x)',       setup="from __main__ import
vec_time = timeit.timeit('exercise22_vectorized(x)', setup="from __main__ import
print("You vectorized code ran %.1fx faster than the original code on a 200x200
```

```
Correct!
You vectorized code ran 642.2x faster than the original code on a 200x200 matrix
```

Exercise 2.3 – Check the quality of a solution to a linear system

Suppose you are given matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$ and vector $\mathbf{b} \in \mathbb{R}^m$ and are told that $\mathbf{x} \in \mathbb{R}^n$ is a solution to the system of linear equations $\mathbf{A}\mathbf{x} = \mathbf{b}$.

Write a Python function that tests whether $\mathbf{x}$ is a solution to $\mathbf{A}\mathbf{x} = \mathbf{b}$. If we denote the m-dimensional vector of residuals as $\mathbf{r} = \mathbf{A}\mathbf{x} - \mathbf{b}$, then your function should return true if and only if *all* residuals $|r_i| < \epsilon$ for a given tolerance parameter ϵ.

In [166…

```
def is_solution(A, b, x, epsilon=1e-5):
    """Returns whether x is a solution to Ax=b, given an absolute tolerance on r
    # Your code here. Aim for 1 line.
    return (np.abs(A @ x) - b < epsilon).all()
```

Check your answer by running the cell below.

In [167…

```
assert 'is_solution' in globals(), "You forgot to run the code cell with your an
A = np.array([[2., 0.5], [-5., 3.]])
b = np.array([5., 9.])
x = np.array([1.23529, 5.05882])
result = is_solution(A, b, x)
assert isinstance(result, np.bool_), "Wrong result type! Expected single Numpy b
assert result, "Wrong answer! x is a solution within tolerance 1e-5"
result = is_solution(A, b, x, epsilon=1e-8)
assert isinstance(result, np.bool_), "Wrong result type! Expected single Numpy b
assert not result, "Wrong answer! x is not a solution within tolerance 1e-8"
print("Correct!")
```

```
Correct!
```

Exercise 2.4 – Use random numbers to shuffle a pair of matrices together

Suppose you are given a pair of matrices $X \in \mathbb{R}^{m \times n}$ and $Y \in \mathbb{R}^{m \times k}$, and you must 'shuffle' their rows by the same permutation.

For example, consider the pair

$$X = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix}, Y = \begin{bmatrix} 10 \\ 20 \\ 30 \end{bmatrix}.$$

The following is a valid row-wise shuffle where new rows $(0, 1, 2)$ are taken from original rows $(1, 2, 0)$

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 1 \\ 0 & 0 \end{bmatrix}, Y = \begin{bmatrix} 20 \\ 30 \\ 10 \end{bmatrix}$$

whereas the following is an *invalid* row-wise shuffle because X and Y are not formed by the same permutation (their rows no longer match up)

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 1 \\ 0 & 0 \end{bmatrix}, Y = \begin{bmatrix} 30 \\ 20 \\ 10 \end{bmatrix}.$$

Write a function that returns a new pair X and Y that are row-shuffled versions of the original arguments. Use Numpy's permutation function. The rows of both X and Y should be shuffled by the same permutation, *not* shuffled independently. Do not modify the original ndarray objects. If X and Y do not have the same number of rows, raise a *ValueError* exception. Aim for 4 lines of code.

In [168…

```
def shuffle_dataset(X, Y):
    """Returns a pair of new ndarrays X, Y where the rows have been shuffled by

    X and Y must refer to two ndarrays that have the same number of rows.
    Does not modify the original X and Y ndarray objects.
    """
    # Your code here
    if len(X) != len(Y):
        raise ValueError
    permutation = np.random.permutation(len(X))
    return X[permutation], Y[permutation]
```

Check your answer by running the code cell below.

In [169…

```
m, n, k = 4, 3, 2   # Check for X 4x3 and Y 4x2 ndarrays
num_trials = 500    # Should be enough trials to see all (m!) possible shuffles
perms = set()       # Collect all unique permutations we've seen returned by the
X = np.arange(m*n).reshape((m, n))
Y = np.arange(m*k).reshape((m, k))
for i in range(num_trials):
    Xarg = X.copy()
    Yarg = Y.copy()
    Xnew, Ynew = shuffle_dataset(Xarg, Yarg)  # Run the student code
    assert np.array_equal(Xarg, X), "Your code wasn't supposed to modify the X a
    assert np.array_equal(Yarg, Y), "Your code wasn't supposed to modify the Y a
    Xperm = np.argsort(Xnew[:,0])  # Undo the permutation via sorting, since ori
    Yperm = np.argsort(Ynew[:,0])  # array elements were increasing by row
    assert np.array_equal(Xnew[Xperm], X), "Your code didn't return a row permut
    assert np.array_equal(Ynew[Yperm], Y), "Your code didn't return a row permut
    assert np.array_equal(Xperm, Yperm), "Your code didn't return the same row p
    perms.add(tuple(Xperm))  # Count the number of times this permutation was en

if len(perms) != 24:
    print("Warning: observed %d of 24 possible permutations after %d trials." %
else:
    try:
        shuffle_dataset(X[:-1], Y)  # Run the student code with invalid input
    except ValueError:
        print("Looks good!")
    else:
        raise AssertionError("Your code was supposed to raise a ValueError"
                             "if X and Y had different number of rows")
```

```
Looks good!
```

3. Plotting exercises

First, in the code cell below, write code to import the *pyplot* module from the Matplotlib package in the standard way, then run the code cell.

In [170…
```
import matplotlib.pyplot as plt
```

If you imported the *pyplot* module from Matplotlib correctly you should be able to run the code cell below without error.

In [171…
```
assert 'matplotlib' not in globals(), "You didn't need to import matplotlib itse
assert 'pyplot' not in globals(), "You didn't import pyplot in the standard way.
assert 'plt' in globals(), "You didn't import pyplot in the standard way. Do Ker
print("Ready!")
```

```
Ready!
```

Exercise 3.1 – Plot a function

Plot these functions over the interval $x \in [-5, 5]$:

1. The sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}}$
2. The rectifier) function $f(x) = \max(0, x)$

Requirements:

- Use Numpy ndarrays and Numpy functions for your solution, *not* Python lists.
- Plot both curves in the same plot
- Use the **plt.ylim** function to set the y axis range to $[-0.1, 3.1]$.
- Use the **plt.grid** function to show a grid.
- Use the **plt.xlabel** function to set the x-axis label.
- Use the **plt.legend** function to add a legend.
- Use the **plt.title** function to add a title.

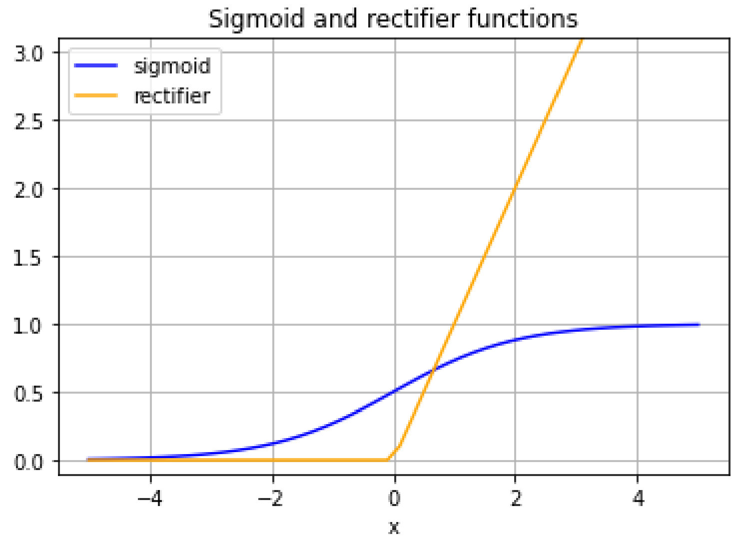
You should be able to reproduce the plot below:

In [172…
```
# Your code here


x = np.linspace(-5, 5)
y = 1/(1 + np.exp(-x))
z = np.maximum(0, x)

plt.plot(x, y, 'b', label='sigmoid')
plt.plot(x, z, 'orange', label='rectifier')
plt.ylim(-0.1, 3.1)
plt.grid()
plt.xlabel('x')
plt.legend()
plt.title('Sigmoid and rectifier functions')

plt.show()
```



Exercise 3.2 – Plot accuracy of finite differences

Plot the accuracy of the central difference approximation from Exercise 1.4 for different values of the step size ϵ. Your plot should show that the finite difference approximation breaks down for values of ϵ that are too large or too small.

Run the code cell below to define functions for $f(x) = x^3$ and its exact first derivative $f'(x) = 3x^2$.

In [173…

```
def f(x):
    return x**3

def df_exact(x):
    return 3 * x**2
```

First, plot the `f` and `f_exact` functions over range $x \in [-1.2, 1.2]$, including title and legend as shown below:

Second, once you've got the above plot, use your `approx_derivative_of(f, epsilon)` function from Exercise 1.4 to add three new series to your plot, corresponding to $\epsilon = 1$, $\epsilon = 10^{-8}$ and $\epsilon = 10^{-16}$. Your new plot should look like this:

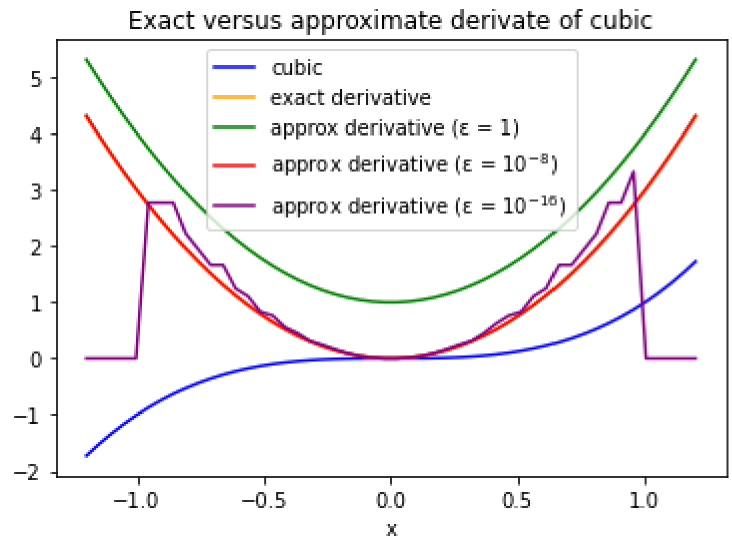
Make greek symbols like ϵ in a label or a title by using the dollar sign ($) and superscript (^), such as `"Argument $\beta$ has value $10^3$"` would render in the plot as Argument β has value 10^3

In [174…

```
# Your code here. Aim for 8-12 lines.
x = np.linspace(-1.2, 1.2)

plt.plot(x, f(x), 'b', label='cubic')
plt.plot(x, df_exact(x), 'orange', label='exact derivative')
plt.plot(x, approx_derivative_of(f, 1)(x), 'g', label='approx derivative (\u03B5
plt.plot(x, approx_derivative_of(f, 1e-8)(x), 'r', label='approx derivative (\u0
```

```
plt.plot(x, approx_derivative_of(f, 1e-16)(x), 'purple', label='approx derivativ
plt.xlabel('x')
plt.title('Exact versus approximate derivate of cubic')
plt.legend()
plt.show()
```



Exercise 3.3 – Plot a stack of images

Stack two images and plot them. Follow these steps:

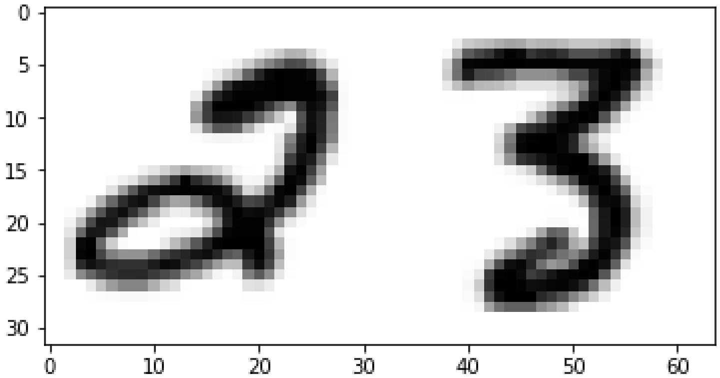
1. The lab comes with a file called mnist-digit-2.png. It contains a 32x32 RBG image (red, green, blue colour channels).
2. Use the **plt.imread** function the load the file. Notice that the result of *imread* is an *ndarray* object of dtype *np.uint8* and shape (32,32,3).
3. Use the **plt.imshow** function to plot the image. The plot should look like this:
4. The lab also comes with a file called mnist-digit-3.png. It contains another 32x32 RGB image.
5. Use the **np.hstack** function to create a new image where the digits are side-by-side.
6. Plot new composite image. It should look like this:

In [175…

```
# Your code here. Aim for 3-4 lines.
img1 = plt.imread('mnist-digit-2.png')
img2 = plt.imread('mnist-digit-3.png')
img = np.hstack((img1, img2))
plt.imshow(img)
```

Out[175…

```
<matplotlib.image.AxesImage at 0x7fdf46f9fd90>
```



In [ ]:

In [ ]: