

Outline

- Python Programming Setup
- Python Platforms for DL
- Introduction to Numpy
- Plotting with Matplotlib
- Preparing Data for Machine Learning
- Lab Assignment

Part 1: Python Programming Setup

Setting up Python Environment (Anaconda 3)

What is Anaconda?



Anaconda is a distribution of the Python and R for scientific computing

- Comes with >250 packages automatically installed
- >7500 additional open-source packages available
- Equipped with Jupyter Notebook
- Conda environment manager for easy maintenance of packages

Setting up Python Environment (Anaconda 3)

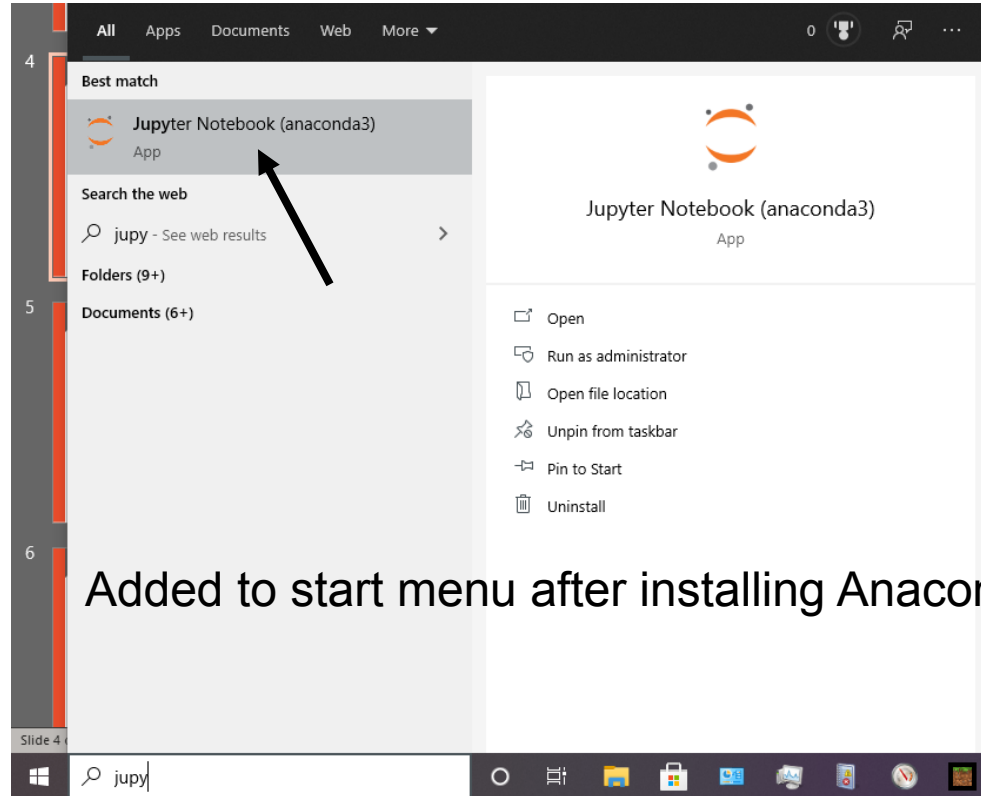
Installing Anaconda 3 <https://www.anaconda.com/products/individual>

Anaconda Installers

Windows 	MacOS 	Linux 
Python 3.8 64-Bit Graphical Installer (457 MB) 32-Bit Graphical Installer (403 MB)	Python 3.8 64-Bit Graphical Installer (435 MB) 64-Bit Command Line Installer (428 MB)	Python 3.8 64-Bit (x86) Installer (529 MB) 64-Bit (Power8 and Power9) Installer (279 MB)

Starting up Jupyter Notebook (Anaconda3)

Windows

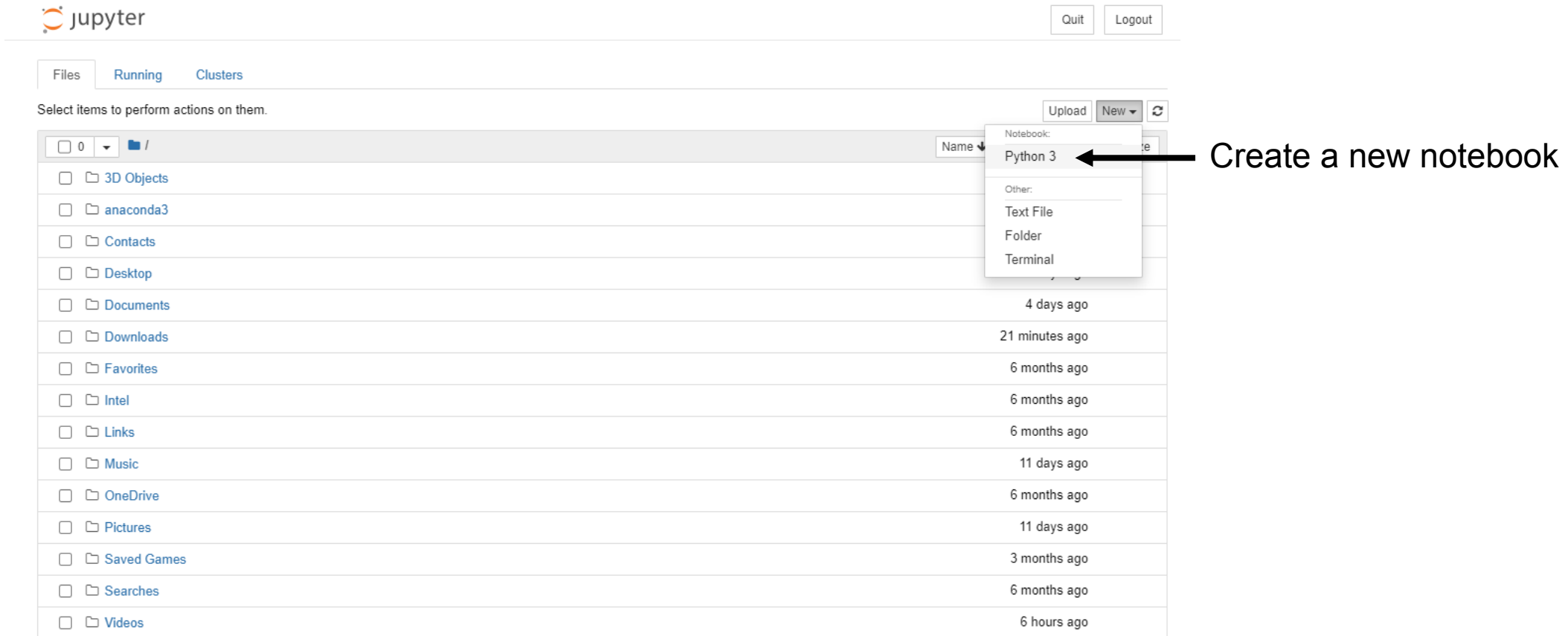


Added to start menu after installing Anaconda 3

Mac/Linux

Start terminal
Type “jupyter notebook”

Starting up Jupyter Notebook (Anaconda3)



The screenshot shows the Jupyter Notebook interface. At the top, there's a header with the Jupyter logo and 'Quit' and 'Logout' buttons. Below the header, there are tabs for 'Files', 'Running', and 'Clusters'. The 'Files' tab is active, showing a file browser. The file browser has a search bar and a list of files and folders. A dropdown menu is open, showing options for creating a new notebook, text file, folder, or terminal. An arrow points to the 'Python 3' option in the dropdown menu, with the text 'Create a new notebook' next to it.

Jupyter

Quit Logout

Files Running Clusters

Select items to perform actions on them.

Upload New

0 /

Name

Notebook:

Python 3

Other:

Text File

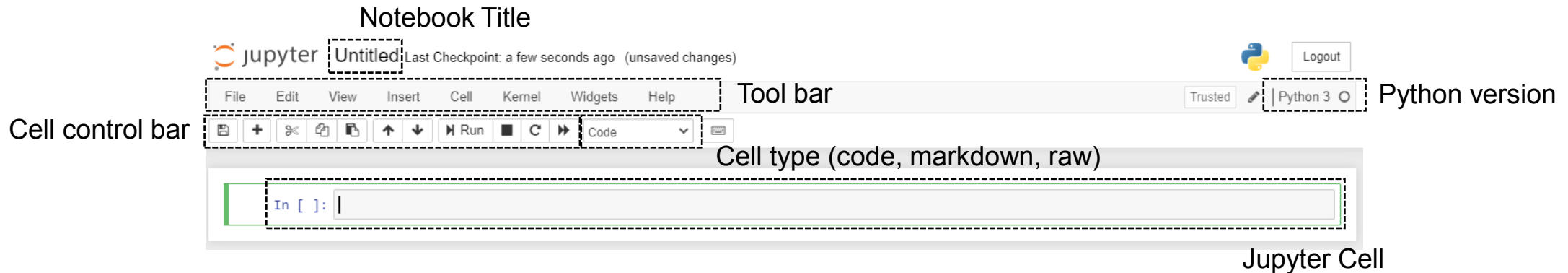
Folder

Terminal

Create a new notebook

Name	Modified
3D Objects	
anaconda3	
Contacts	
Desktop	
Documents	4 days ago
Downloads	21 minutes ago
Favorites	6 months ago
Intel	6 months ago
Links	6 months ago
Music	11 days ago
OneDrive	6 months ago
Pictures	11 days ago
Saved Games	3 months ago
Searches	6 months ago
Videos	6 hours ago

Starting up Jupyter Notebook (Anaconda3)



For more information, visit <https://www.dataquest.io/blog/jupyter-notebook-tutorial/>

Part 2: Python Platforms for DL

Google Colaboratory

A free Jupyter notebook environment that runs in the cloud

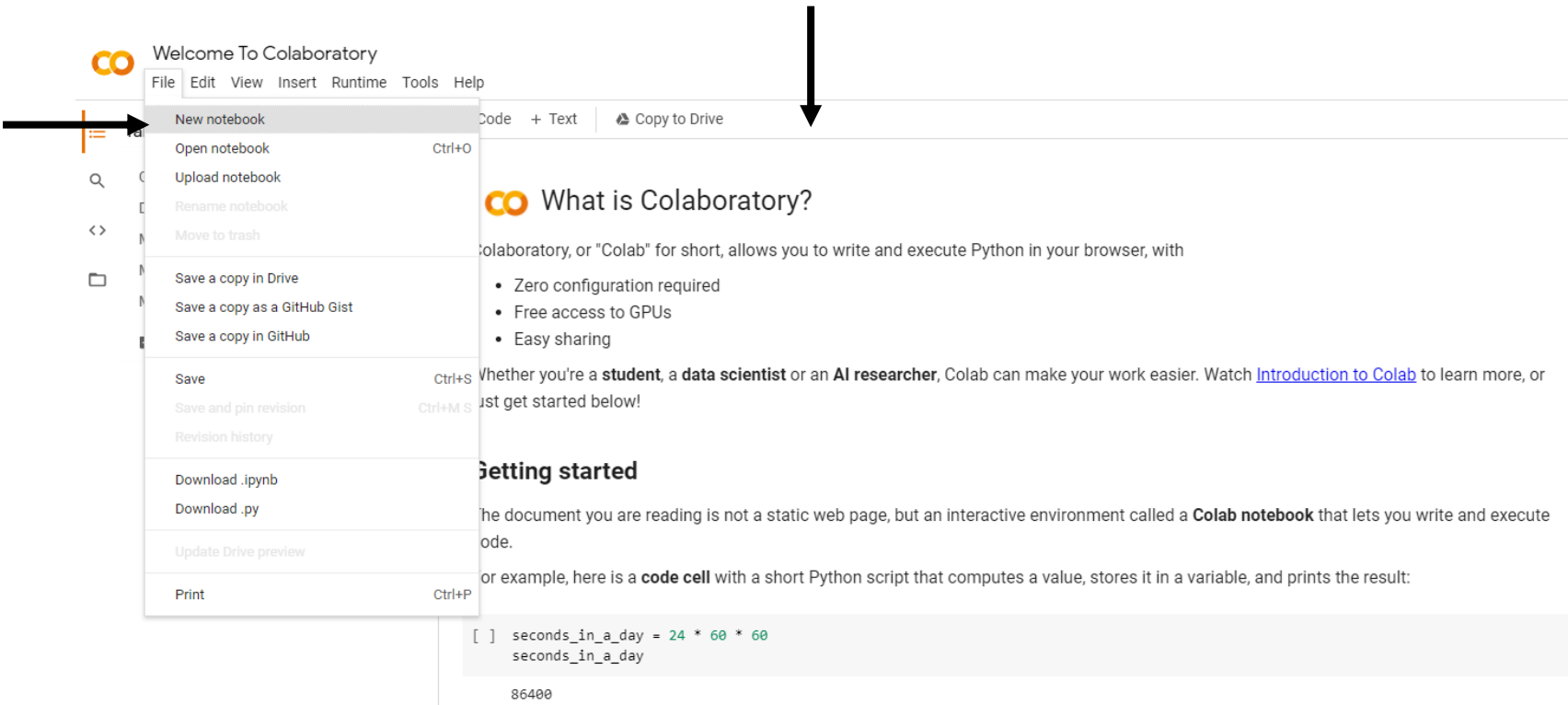
- Saves in Google drive
- Github commit style code sharing with others
- Maximum runtime of 12hrs (Free version)
- Support GPU or TPU for hardware acceleration
- Pre-equipped with latest scientific packages (Numpy, Scipy, etc)



Google Colaboratory: Getting started

Tutorial to Colab <https://colab.research.google.com/notebooks/intro.ipynb>

Create new
Notebook



The screenshot shows the Google Colaboratory web interface. On the left, a sidebar contains icons for file management. The 'File' menu is open, and the 'New notebook' option is highlighted. A black arrow points from the text 'Create new Notebook' to this menu item. Another black arrow points from the URL 'https://colab.research.google.com/notebooks/intro.ipynb' to the top of the interface. The main content area displays the 'Welcome To Colaboratory' page, which includes a 'What is Colaboratory?' section with bullet points: 'Zero configuration required', 'Free access to GPUs', and 'Easy sharing'. Below this is a 'Getting started' section with a code cell containing a Python script that calculates the number of seconds in a day (24 * 60 * 60), resulting in 86400.

Welcome To Colaboratory

File Edit View Insert Runtime Tools Help

New notebook
Open notebook Ctrl+O
Upload notebook
Rename notebook
Move to trash
Save a copy in Drive
Save a copy as a GitHub Gist
Save a copy in GitHub
Save Ctrl+S
Save and pin revision Ctrl+M S
Revision history
Download .ipynb
Download .py
Update Drive preview
Print Ctrl+P

What is Colaboratory?

Colaboratory, or "Colab" for short, allows you to write and execute Python in your browser, with

- Zero configuration required
- Free access to GPUs
- Easy sharing

Whether you're a **student**, a **data scientist** or an **AI researcher**, Colab can make your work easier. Watch [Introduction to Colab](#) to learn more, or just get started below!

Getting started

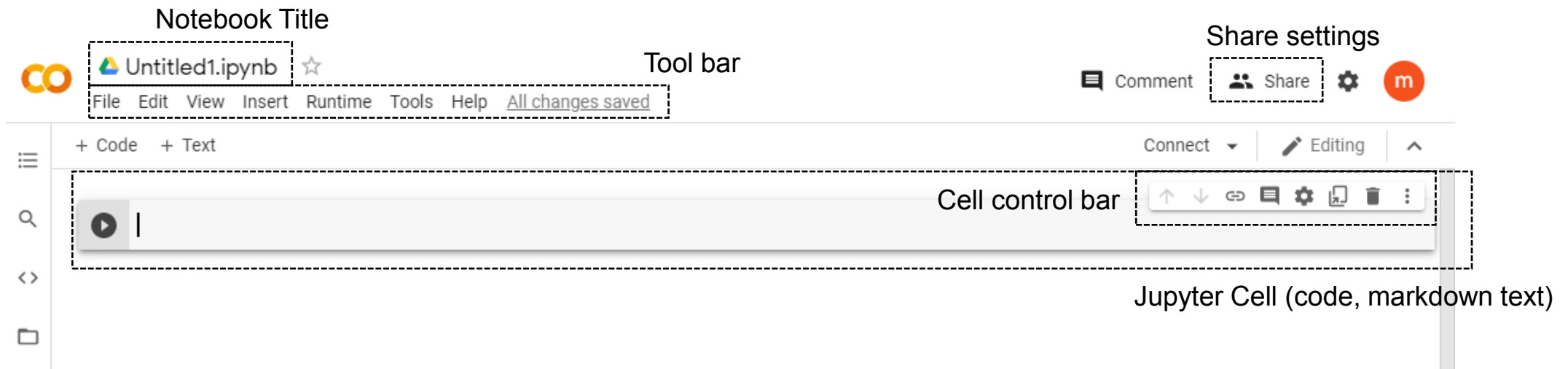
The document you are reading is not a static web page, but an interactive environment called a **Colab notebook** that lets you write and execute code.

For example, here is a **code cell** with a short Python script that computes a value, stores it in a variable, and prints the result:

```
[ ] seconds_in_a_day = 24 * 60 * 60
seconds_in_a_day

86400
```

Google Colaboratory: Getting started



Google Cloud

Suite of cloud computing services offered by Google

- Offers AI Platform for deploying DL models
- Support Jupyter Notebook instances
- Provide instances with DL libraries
- Fully customizable hardware spec with state-of-the-art components
- Monthly charge for the service



Setup tutorial by “Towards data science” → <https://towardsdatascience.com/get-deep-learning-on-google-cloud-platform-the-easy-way-53f74bab5ee9>

Deep Learning Frameworks



Developed by Facebook

- Lots of modules that are easy to combine
- Easy to edit network
- Lots of pre-trained models
- Seamless integration into Python/Numpy framework



TensorFlow

Developed by Google

- Provides Tensorboard for visualization
- Uses its own session during training
- Great community support
- Tensorflow Lite can run models on mobile devices



Developed by Apache

- Supported by Amazon Web Service
- Supports many languages
- Fast and flexible for running DL algorithms
- Features advanced GPU support
- Popular among industrial projects

Deep Learning Frameworks



Developed by Facebook

- Lots of modules that are easy to combine
- Easy to edit network
- Lots of pre-trained models
- Seamless integration into Python/Numpy framework



TensorFlow

Developed by Google

- Provides Tensorboard for visualization
- Uses its own session during training
- Great community support
- Tensorflow Lite can run models on mobile devices



Developed by Apache

- Supported by Amazon Web Service
- Supports many languages
- Fast and flexible for running DL algorithms
- Features advanced GPU support
- Popular among industrial projects

Framework for this class

Part 3: Introduction to Numpy

What is Numpy?

Fundamental package for scientific computing in Python

- Provides multi-dimensional array object
- Provides assortment of mathematical routines for arrays
- Fast array operations through pre-compiled C
- Support vectorization of operations
- Seamlessly integrated with DL frameworks such as PyTorch, TensorFlow



Constructing Numpy arrays

From python lists

```
# Defining a numpy array

# 1D array
arr = np.array([1,2,3,4,5])

# 2D array
arr_2d = np.array([[1,2,3,4,5],
                  [6,7,8,9,10],
                  [11,12,13,14,15]])

# 3D array
arr_3d = np.array([[1,2,3,4,5],
                  [6,7,8,9,10],
                  [11,12,13,14,15]],
                  [[16,17,18,19,20],
                  [21,22,23,24,25],
                  [26,27,28,29,30]])
```

From Numpy commands

```
# Define number of each dimension

n1 = 3
n2 = 4
n3 = 5

# Zeros array
zeros_1d = np.zeros(n1)
zeros_2d = np.zeros((n1,n2))
zeros_3d = np.zeros((n1,n2,n3))

# Ones array
ones_1d = np.ones(n1)
ones_2d = np.ones((n1,n2))
ones_3d = np.ones((n1,n2,n3))

# Creating array using np.arange
arr_arange = np.arange(0, 10, 1) # (start, stop, stepsize)

# Creating an array using np.linspace
arr_linspace = np.linspace(0, 9, 10) # (start, stop, # of bins)
```

Random arrays

```
# Random array

np.random.seed(10) # Fixes the seed number so that random samplings always give same results

rand_arr = np.random.randn(n1, n2) # Random array sampled from standard normal distribution
```

Basic Matrix Operations in Numpy

Elementwise Addition – `np.add()`

Dot Product – `np.dot()`

Elementwise Subtraction – `np.subtract()`

Transpose – `.T` operative or `np.transpose()`

Elementwise Multiplication – `np.multiply()`

Elementwise Division – `np.divide()`

Elementwise Power – `np.power()`

Useful Numpy functions

Combining arrays

Concatenating arrays – `np.concetenate()`

Stacking arrays – `np.stack()` (Can add dimensions), `np.hstack()` (horizontal stack), `np.vstack()` (vertical stack)

Finding characteristic values of an array

Minimum, Maximum, Mean, Sum of array elements – `np.min()`, `np.max()`, `np.mean()`, `np.sum()`

Indexing an array

Indices of minimum and maximum element – `np.argmin()`, `np.argmax()`

Sorting the indices from low to high values – `np.argsort()`

Finding the indices that satisfy conditions – `np.where()`

Part 4: Plotting with Matplotlib

Basic plotting

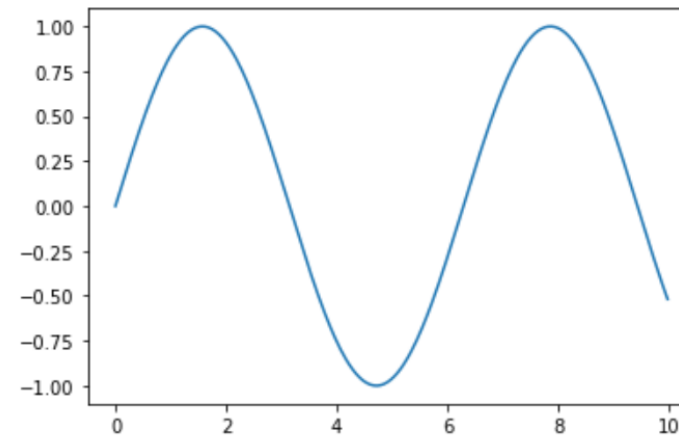
Import Matplotlib

```
##matplotlib inline # If using local notebook runtime, allows you to display the plot inside the jupyter notebook  
##matplotlib notebook # Alternatively, you can use this line instead for interactive plots  
  
import matplotlib.pyplot as plt
```

Code

```
x = np.arange(0, 10, 1/32) # x axis data  
y = np.sin(x)             # y axis data  
plt.plot(x, y)            # plot the data
```

Output

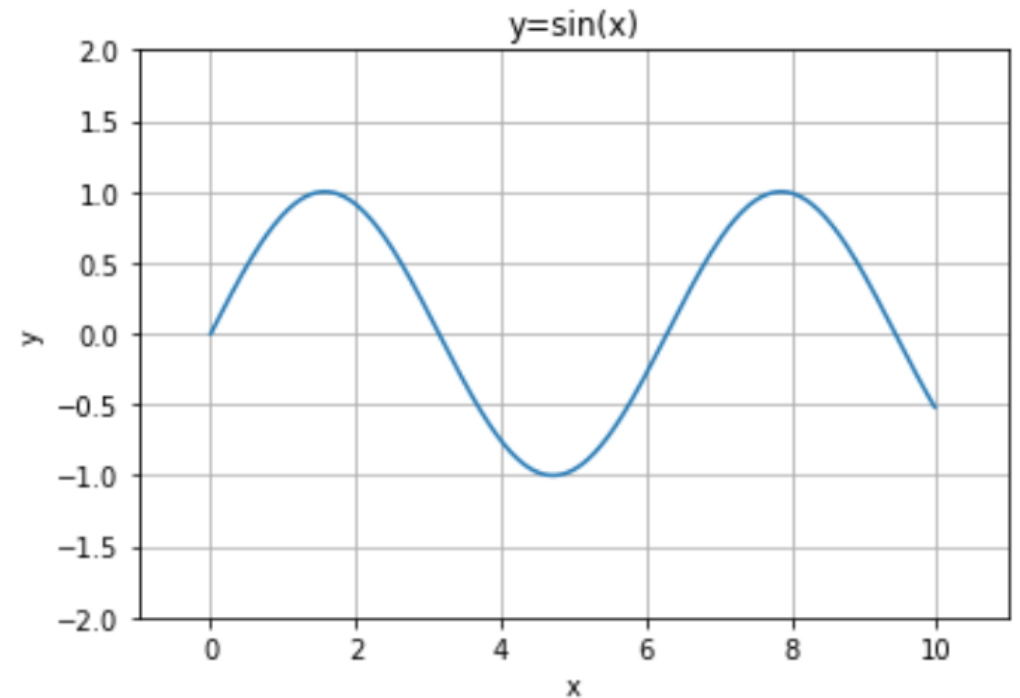


Labeling your plots

Code

```
plt.plot(x, y)
plt.title('y=sin(x)') # set the title
plt.xlabel('x')       # set the x axis label
plt.ylabel('y')       # set the y axis label
plt.xlim(-1, 11)     # set the x axis range
plt.ylim(-2, 2)      # set the y axis range
plt.grid()            # enable the grid
```

Output

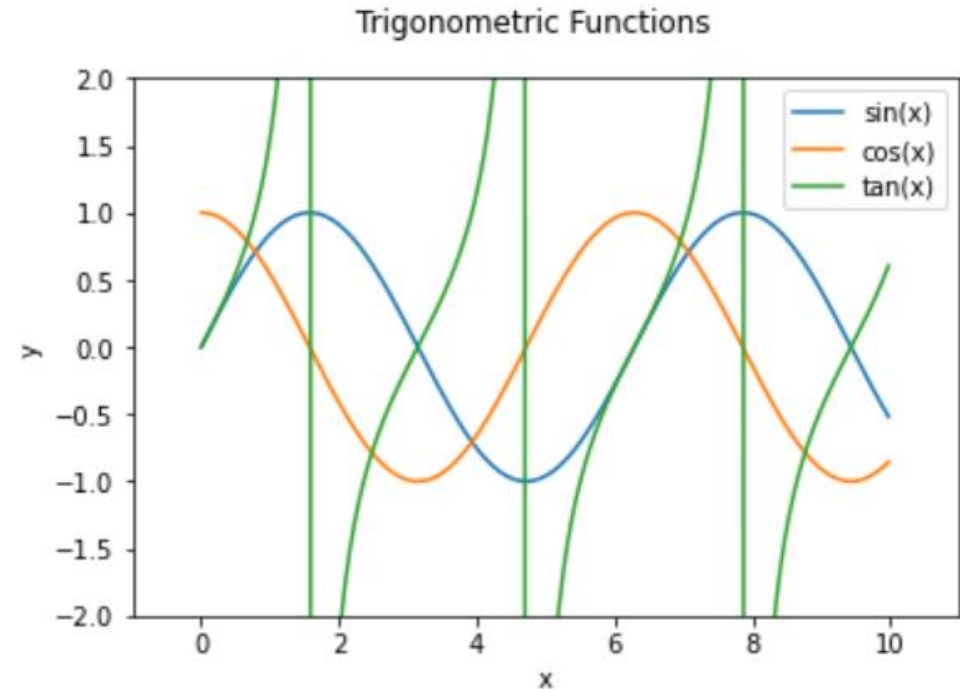


Multiple plots

Code

```
# Multiple Plots
# On same figure
x = np.arange(0, 10, 1/32) # x axis data
y1 = np.sin(x)             # y axis data 1
y2 = np.cos(x)             # y axis data 2
y3 = np.tan(x)             # y axis data 3
plt.figure(1)              # create figure 1
plt.plot(x, y1, label='sin(x)')
plt.plot(x, y2, label='cos(x)')
plt.plot(x, y3, label='tan(x)')
plt.xlabel('x')
plt.ylabel('y')
plt.xlim(-1, 11)
plt.ylim(-2, 2)
plt.suptitle('Trigonometric Functions')
plt.legend()
plt.show()
```

Output



Creating subplots

Code

```
# Multiple Subplots
x = np.arange(0, 10, 1/32) # x axis data
y1 = np.sin(x)             # y axis data for subplot 1
y2 = np.cos(x)             # y axis data for subplot 2
y3 = np.tan(x)             # y axis data for subplot 3

fig = plt.figure(2, figsize=(8,8)) # create figure 2

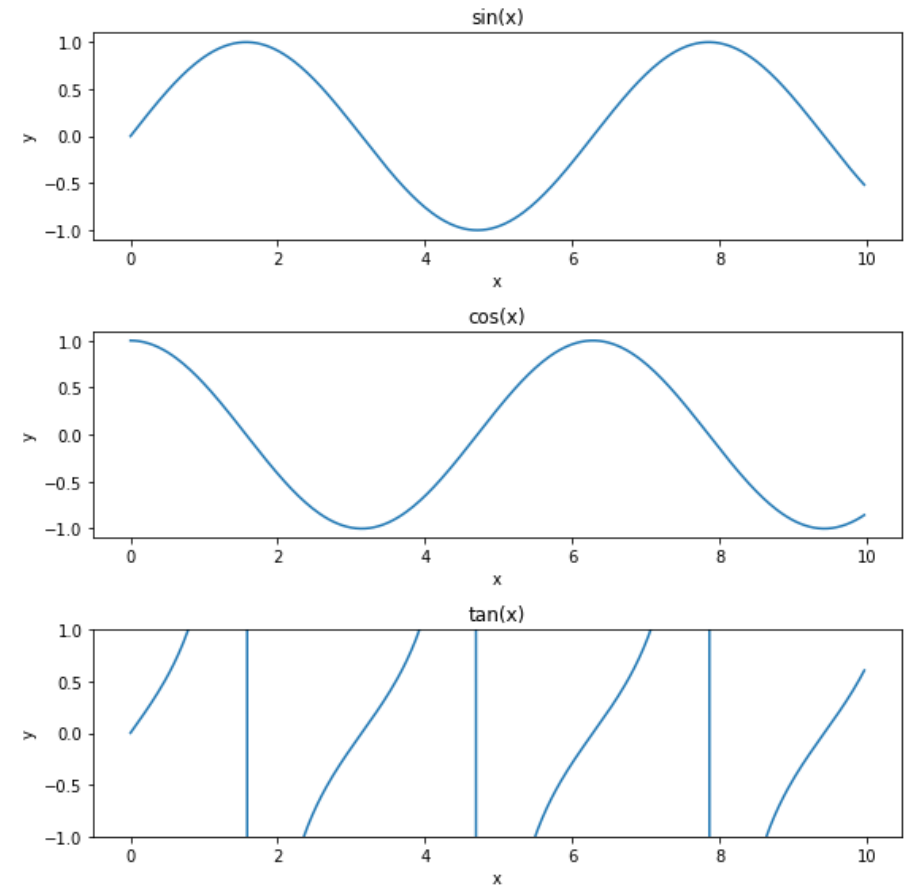
plt.subplot(311)            # (number of rows, number of columns, current plot)
plt.plot(x, y1)
plt.title('sin(x)')
plt.xlabel('x')
plt.ylabel('y')

plt.subplot(312)
plt.plot(x, y2)
plt.title('cos(x)')
plt.xlabel('x')
plt.ylabel('y')

plt.subplot(313)
plt.plot(x, y3)
plt.title('tan(x)')
plt.xlabel('x')
plt.ylabel('y')
plt.ylim(-1, 1)

fig.tight_layout()
```

Output



Part 5: Working with data

Loading dataset

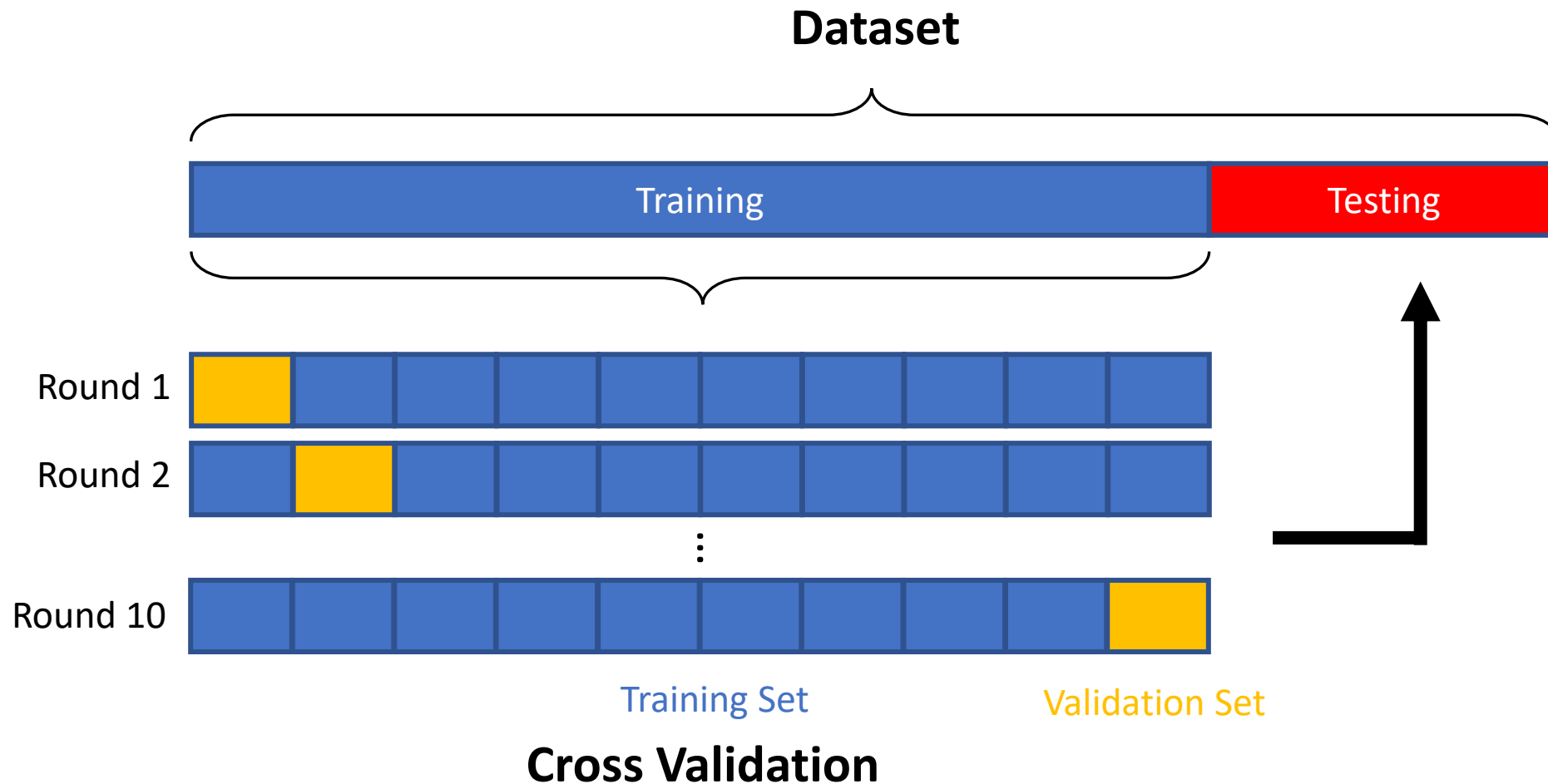
```
import pandas as pd
import sklearn

# Import necessary modules
from sklearn.linear_model import LogisticRegressionCV

diabetes = pd.read_csv('diabetes.csv') # Read the dataset with pandas
diabetes.head() # Display the head of the data
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

Dividing dataset into Train, Validation, Test



Logistic Regression using Scikit-Learn

Scaling the dataset

Scaling the data

```
X1 = diabetes.values[:, :-1]          # Extract features
Y1 = diabetes.values[:, -1]          # Extract labels
```

Scale the data $z = (x - u) / s$

```
X1_mean = np.mean(X1, axis = 0)      # Compute means
X1_std = np.std(X1, axis = 0)        # Compute stds
X1_mean_repeated = np.tile(X1_mean, (len(X1), 1)) # Create repeated array of mean to match X1 dimension
X1_std_repeated = np.tile(X1_std, (len(X1), 1))  # Create repeated array of std to match X1 dimension
X1_scaled = np.divide(np.subtract(X1, X1_mean_repeated), X1_std_repeated) # Compute z for the array
```

Divide the data into training and testing

```
test_ratio = 0.3                     # Set the test data ratio
train_size = int(len(X1)*(1-test_ratio)) # Set training data size based on test ratio
```

```
X_train = X1_scaled[:train_size]
X_test = X1_scaled[train_size:]
Y_train = Y1[:train_size]
Y_test = Y1[train_size:]
```

Use cross validation to train

```
model = LogisticRegressionCV(cv=10).fit(X_train, Y_train)
result = model.score(X_test, Y_test)
```

```
print("Accuracy: %.2f%%" % (result*100))
```

Accuracy: 78.79%

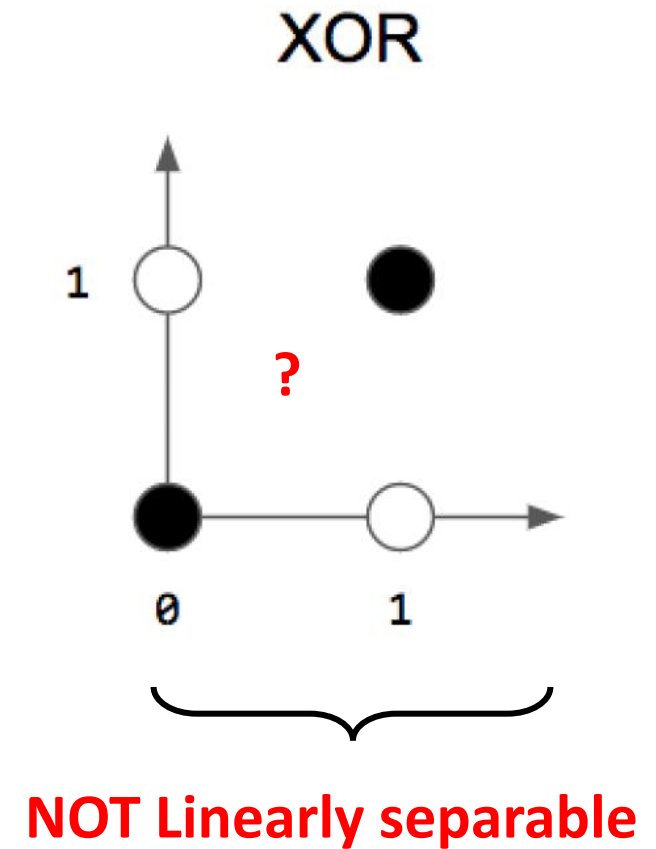
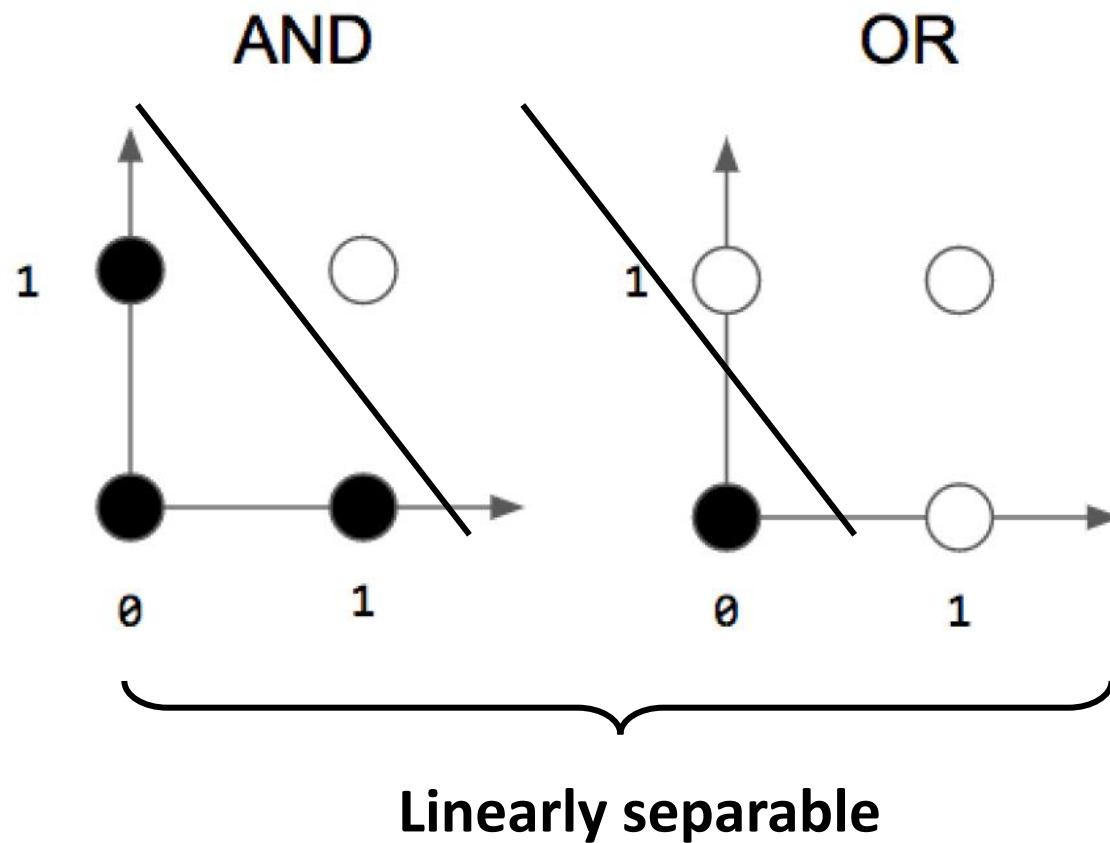
Define Training and Testing sets

Perform Logistic Validation with CV

Lab Assignment:

Implement Neural Network for XOR gate
with Numpy

XOR Problem



Loading dataset into Numpy array

Using Pandas

```
# XOR table

# Using Pandas
import pandas as pd

XOR_table = pd.read_csv('XOR_table.csv')
XOR_table
```

	x1	x2	y
0	0	0	0
1	0	1	1
2	1	0	1
3	1	1	0

Converting data into Numpy array

```
XOR_table = XOR_table.values
X = XOR_table[:, :2]
targets = XOR_table[:, -1].reshape(-1,1)

print(X)          # Input data
print(targets)    # Output targets
```

```
[[0 0]
 [0 1]
 [1 0]
 [1 1]]
[[0]
 [1]
 [1]
 [0]]
```

Solving XOR with a neural network

```
# Define dimensions on input, hidden and output layers
input_dim, hidden_dim, output_dim =

# Define learning rate
learning_rate=

# Define a hidden layer
W1=
# Define an output layer
W2=
# Define sigmoid activation function

for i in range(10000):

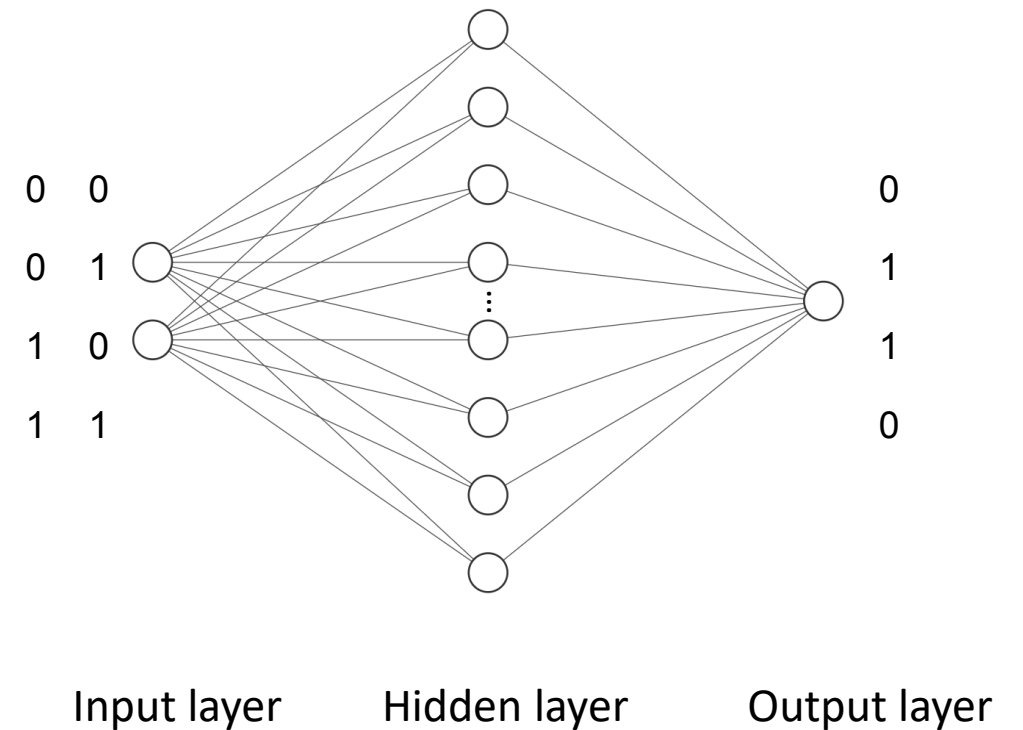
    # Forward pass: compute predicted y

    # Compute and print loss

    # Backprop to compute gradients of w1 and w2 with respect to L2-norm loss

    # Update weights

    # Save loss to an array
```



Solving XOR with a neural network (Forward Pass)

Feed Forward Eqns

$$z = \sigma(W_1x + b_1)$$

$$y = \sigma(W_2\sigma(W_1x + b_1) + b_2)$$

$$y = \sigma(W_2z + b_2)$$

Activation (sigmoid)

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

L2-Loss Function

$$J = \sum_{i=1}^N (y - t)^2$$

y = *predicted output*

t = *Target output*

Solving XOR with a neural network (Backward Pass)

Derivative of Activation

$$\frac{d(\sigma(x))}{dx} = \frac{1}{1 + e^{-x}} * \left(1 - \frac{1}{1 + e^{-x}}\right)$$

$$= \sigma(x)(1 - \sigma(x))$$

Gradient of J w.r.t. W_2

$$\frac{\partial J}{\partial W_2} = \frac{\partial J}{\partial y} \frac{\partial y}{\partial W_2}$$

$$\frac{\partial J}{\partial y} = 2(y - t)$$

$$\frac{\partial y}{\partial W_2} = \sigma(W_2 z + b_2)(1 - \sigma(W_2 z + b_2))z$$

$$\frac{\partial J}{\partial W_2} = 2(y - t)y(1 - y)z$$

Gradient of J w.r.t. W_1

$$\frac{\partial J}{\partial W_1} = \frac{\partial J}{\partial y} \frac{\partial y}{\partial z} \frac{\partial z}{\partial W_1}$$

$$\frac{\partial J}{\partial y} = 2(y - t)$$

$$\frac{\partial y}{\partial z} = \sigma(W_2 z + b_2)(1 - \sigma(W_2 z + b_2))W_2$$

$$\frac{\partial z}{\partial W_1} = \sigma(W_1 x + b_1)(1 - \sigma(W_1 x + b_1))x$$

$$\frac{\partial J}{\partial W_1} = 2(y - t)y(1 - y)W_2 z(1 - z)x$$

Update rule for W_2

$$W_2 = W_2 - \alpha \frac{\partial J}{\partial W_2}$$

$$W_2 = W_2 - \alpha 2(y - t)y(1 - y)z$$

Update rule for W_1

$$W_1 = W_1 - \alpha \frac{\partial J}{\partial W_1}$$

$$W_1 = W_1 - \alpha 2(y - t)y(1 - y)W_2 z(1 - z)x$$