Introduction to AI Networking

Choong Seon Hong

cshong@khu.ac.kr
Department of Computer Science and Engineering
Kyung Hee University, Korea

What is AI Networking

- It is a networking technology for end-to-end connectivity and networking management technology using AI learning algorithms such as :
  - Supervised Learning
  - Unsupervised Learning
  - Reinforcement Learning

- Topics to be covered in the class
  - Edge Computing
  - Federated Learning
  - Resource Management
  - D2D Communication Networks
  - UAV networks
  - Energy Management
  - Security Issues
  - Meta-learning and Transfer Learning

Lecture Schedule and Evaluation

[Schedule]

Week	Contents
1st Week	Introduction to AI Networking
2nd Week	Networking and Future Internet
3rd Week	Mobile Networking(LTE & 5G)
4th Week	Machine Learning based Edge Computing
5th Week	Federated Learning and Democratized Learning
6th Week	AI based Network Resource Management 1
7th Week	AI based Network Resource Management 2
8th Week	AI based D2D Communication Networks
9th Week	Mid-term Exam
10th Week	UAV-Assisted Wireless Networks
11th Week	AI based Energy Management
12th Week	Vehicular Edge Networking
13th Week	Next Generation Security based on Machine Learning
14th Week	Meta-Learning based Networking Architecture
15th Week	6G & AI
16th Week	Term Project Presentations

[Evaluation]

Mid-term Exam: 50%	50%
Presentation & Project	40%
Presence	10%

KHU

The Evolution of Artificial Intelligence (AI)

1950
Alan Turing Proposes the Turing Test

1950
Isaac Asimov proposes the Three Laws of Robotics

1951
First AI based Program was written

1955
First self learning game playing program is written

1959
MIT AI Lab is setup

1961
First Robot is inducted into GM's assembly production line

1963
First Machine Learning program is written

1964
First demonstration of an AI program which understand Natural Language

1965
First AI based Chat-bot (ELLZA) was created

1969
Stanford Research Institute (SRI) demonstrates the first locomotive and intelligent robot (Shakey)

1969
First autonomous vehicle is created at the Stanford AI LAB

1974
First rule based AI expert system for medical diagnostics

1980
LISP based machines are developed and marketed

1986
Learning representations by back-propagating error (Backpropagation) [G. Hinton]

1990
Probabilistic models of sequences (Yoshua Bengio)

1997
IBMs Deep Blue beats Gary Kasparov at Chess

1998
Convolutional Neural Network is introduced by Yann LeCun

1999
First Emotional AI machines demonstrated at MIT AI Lab

2004
DAPRA introduces the first challenge for Autonomous Vehicles

2005
AI based recommendation engines

2009
Google builds Self Driving Car

2010
Narrative Science's AI demonstrates ability to write reports

2011
IBM Watson beats Jeopardy champions

2011
Personal Assistants like Siri, Google Now and Cortana become mainstream

2015
Elon Musk and others announce a $1B non profit open source initiative, OPEN AI to develop friendly AI

2016
Google's Deepmind AlphaGo defeats Go's champions

2016
NVIDIA announces supercomputer for Deep Learning and AI

2017
AlphaGo Zero which learns from scratch

2018
OpenAI 5 (Dota 2)

2019
OpenAI 5 (Defeat the world's top Dota 2 team)

Turing Award

2020
OpenAI's GPT-3

Deep Learning Taxonomy

Deep Learning

Supervised learning

Classification
- Neural Network (NN)
- Convolutional Neural Network (CNN)
- Deep Belief Networks (DBN)
- Recurrent Neural Network (RNN)

Regression
- Neural Network (NN)

Unsupervised Learning

Dimensionality Reduction
- Stacked Auto-Encoders (SAE)
- Auto-Associative Neural Network

Clustering
- Deep Belief Networks (DBN)

Density Estimation
- Deep Boltzmann Machine (DBM)

Reinforcement learning

Reinforcement Learning
- Deep Q-learning
- Double Q-learning
- Prioritized experience replay

Applications

	Applications
Classification	Image classification, Character recognition, Facial recognition, Surveillance systems
Regression	Advertising and business intelligence (Google ads, etc.), Weather forecasting, Market forecasting, Political campaigns
Dimensionality Reduction	Big data visualization, Feature elicitation, Structure discovery, Meaningful compression
Clustering	Recommendation engines (Amazon web service, Netflix, etc.), Customer segmentation, Target marketing, Filter
Density Estimation	Economics (risk prediction, etc.)
Reinforcement Learning	Real-time decisions, Game Artificial Intelligence, Learning tasks, Skill acquisition, Personal assistants (Google Now, Microsoft Cortana, Apple Siri, etc.), Autonomous ("Self-driving") cars

KHU

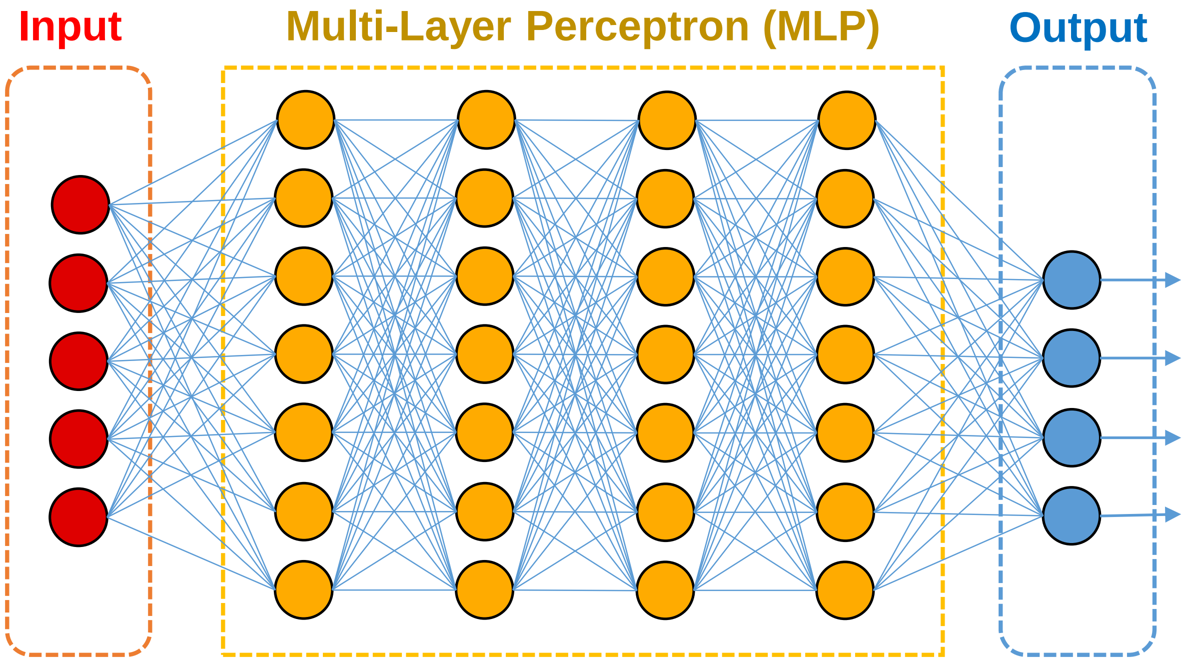
Modified from source: Fadlullah, Zubair, et al. "State-of-the-Art Deep Learning: Evolving Machine Intelligence Toward Tomorrow's Intelligent Network Traffic Control Systems." *IEEE Communications Surveys & Tutorials* (2017).

- How Deep Learning Works?
  - Deep Learning Computation Procedure

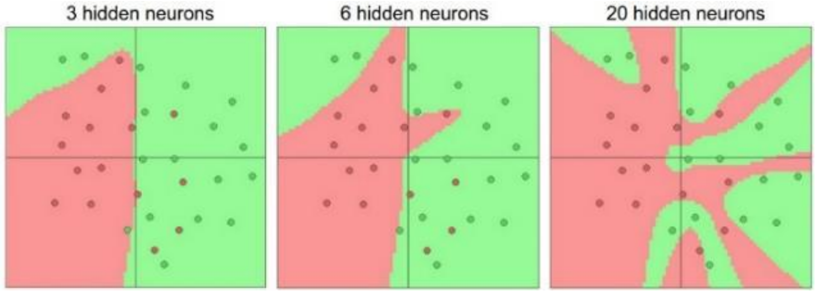
Deep Learning Model Setup
- MLP, CNN, RNN, GAN, or Customized
- # Hidden Layers, # Units, Input/Output, …
- Cost Function / Optimizer Selection

Training (with Large-Scale Dataset)
- Input: Data, Output: Labels
- Learning → Weights Updates for Cost Function Minimization

Inference / Testing (Real-World Execution)
- Input: Real-World Input Data
- Output: Interference Results based on Updated Weights in Deep Neural Networks



Non-Linear Training (Weights Updates) for Cost Minimization: GD, SGD, Adam, etc.



- How Deep Learning Works?
  - Deep Learning Computation Procedure

Deep Learning Model Setup
- MLP, CNN, RNN, GAN, or Customized
- # Hidden Layers, # Units, Input/Output, …
- Cost Function / Optimizer Selection

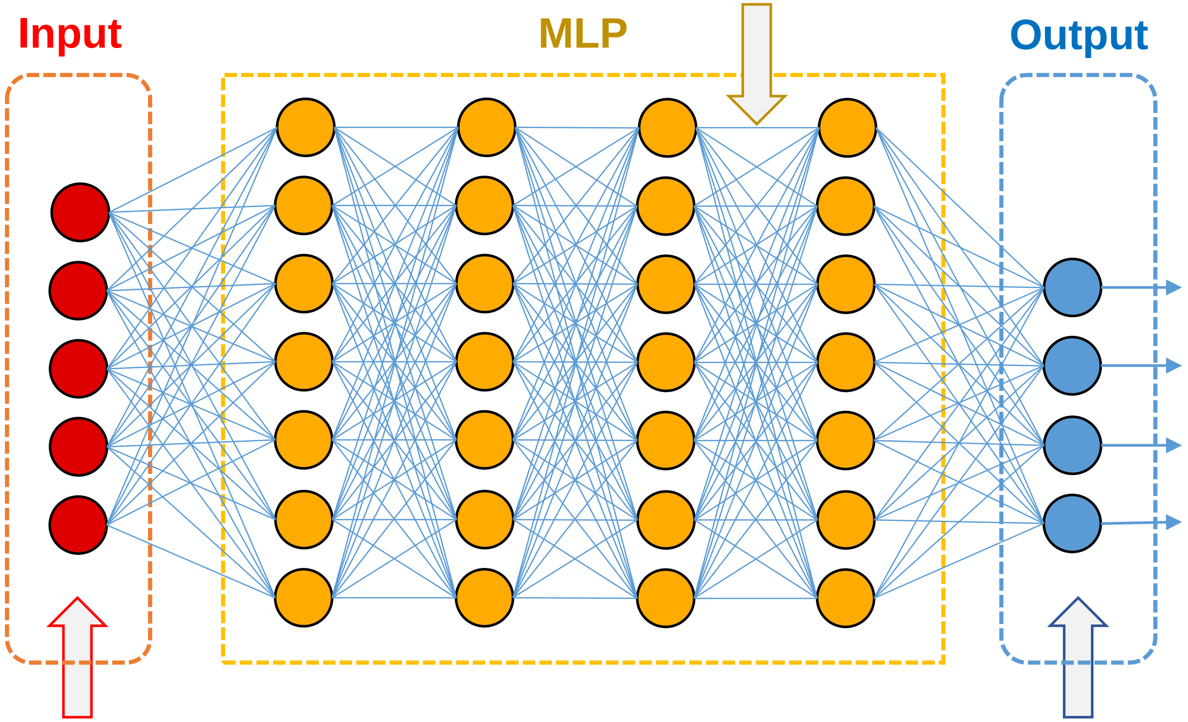
↓

Training (with Large-Scale Dataset)
- Input: Data, Output: Labels
- Learning → Weights Updates for Cost Function Minimization

↓

Inference / Testing (Real-World Execution)
- Input: Real-World Input Data
- Output: Interference Results based on Updated Weights in Deep Neural Networks

All weights in units are trained/set (under cost minimization)



INPUT: Data
- One-Dimension Vector

OUTPUT: Labels
- One-Hot Encoding

We need a lot of training data for generality (otherwise, we will suffer from overfitting problem).

- How Deep Learning Works?
  - Deep Learning Computation Procedure

Deep Learning Model Setup
- MLP, CNN, RNN, GAN, or Customized
- # Hidden Layers, # Units, Input/Output, …
- Cost Function / Optimizer Selection

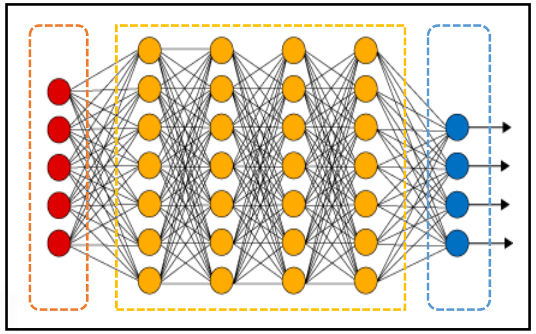
↓

Training (with Large-Scale Dataset)
- Input: Data, Output: Labels
- Learning → Weights Updates for Cost Function Minimization

↓

Inference / Testing (Real-World Execution)
- Input: Real-World Input Data
- Output: Interference Results based on Updated Weights in Deep Neural Networks

Trained Model



Intelligent Surveillance Platforms

INPUT: Real-Time Arrivals

OUTPUT: Inference
- Computation Results based on (i) INPUT and (ii) trained weights in units (trained model).

- How Deep Learning Works?
  - Issue - **Overfitting**

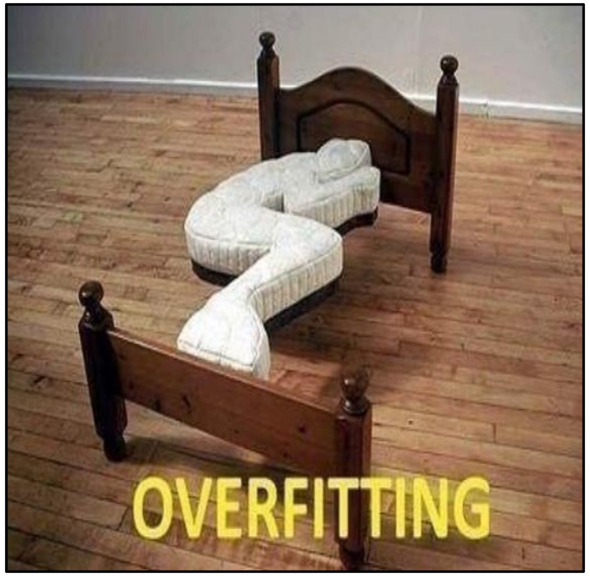
Deep Learning Model Setup
- MLP, CNN, RNN, GAN, or Customized
- # Hidden Layers, # Units, Input/Output, …
- Cost Function / Optimizer Selection

Training (with Large-Scale Dataset)
- Input: Data, Output: Labels
- Learning → Weights Updates for Cost Function Minimization

Inference / Testing (Real-World Execution)
- Input: Real-World Input Data
- Output: Interference Results based on Updated Weights in Deep Neural Networks

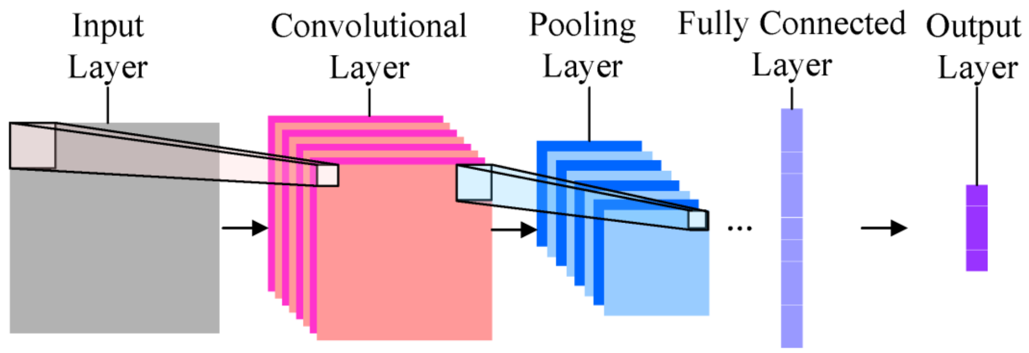
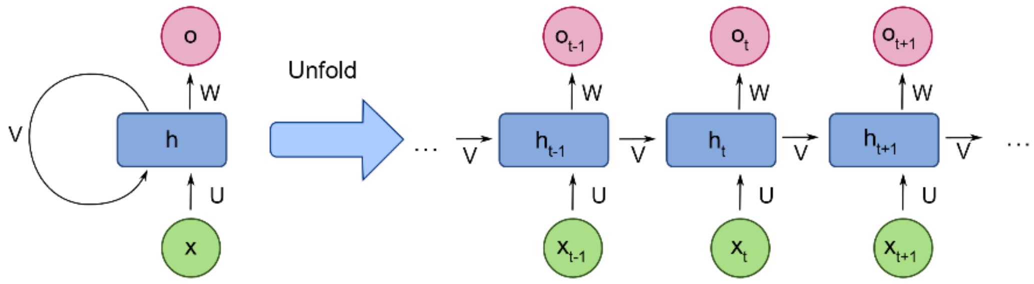
What if we do not have enough data for training (not enough to derive Gaussian/normal distribution)?

Situation becomes worse when the model (with insufficient training data) accurately fits on training data.



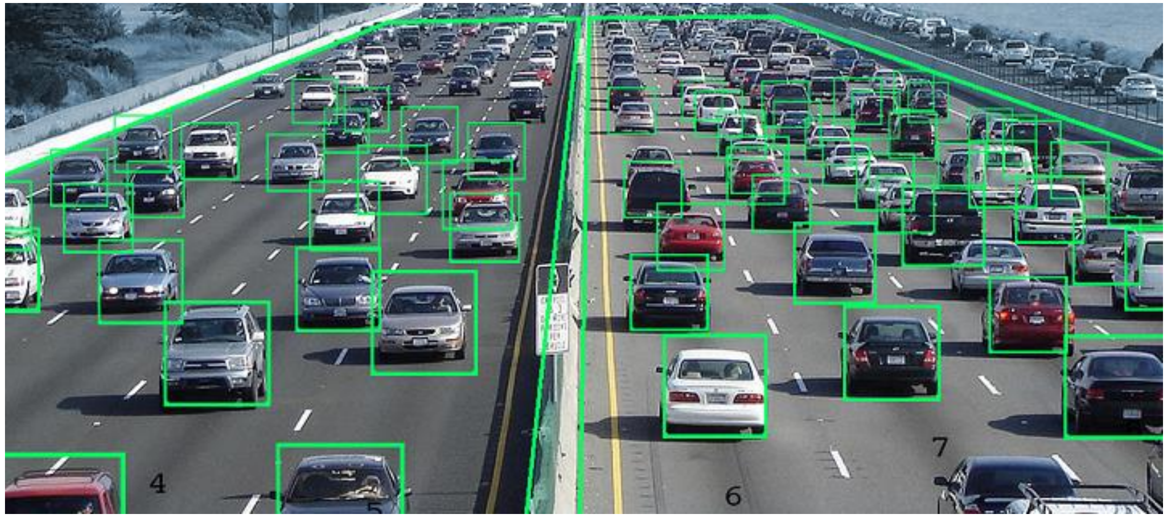
To Combat the Overfitting
- More training data
- Autoencoding (or variational auto-encoder (VAE))
- Droupout
- Regularization

- Two Major Deep Learning Models → CNN vs. RNN

Convolutional Neural Network (CNN)	Recurrent Neural Network (RNN)
	
• In conventional neural network architectures, the input should be one-dimensional vector. • In many applications, the input should be multi-dimensional (e.g., 2D for images). Thus, we need architectures in order to recognize the features in high-dimensional data. • Mainly used for **visual information learning**	• In conventional neural network architectures, there is no way to introduce the concept of time. • The time index can be represented as the chain of neural network models. • The representative models are LSTM and GRU. • Mainly used for **time-series information learning**

Deep Learning: Vision

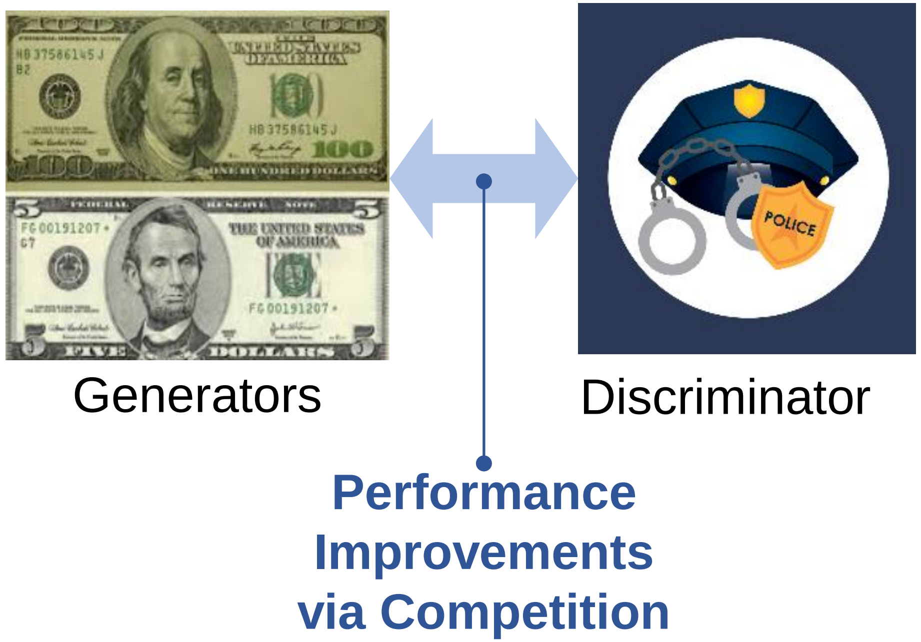
Visual Learning
- Object Recognition
- Style Transfer
- Deblurring and Denoising
- Super-Resolution
- …



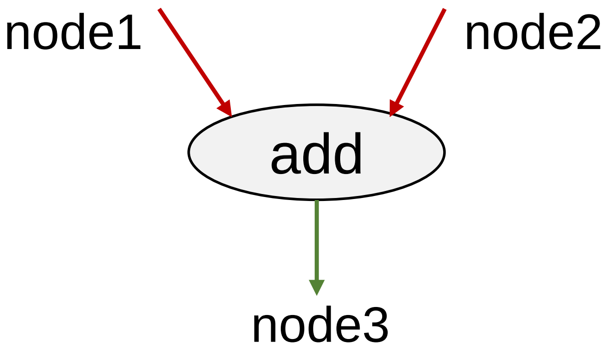
Speech/Language Learning

- Speech Recognition
- Machine Translation
- Information Retrieval
- …

- An Emerging Direction, Generative Adversarial Network (GAN)
  - Training both of **generator** and **discriminator**; and then generates samples which are similar to the original samples.



- Quick Start Example



- **add** is a node which represents addition operation
  - **node1**: input tensor
  - **node2**: input tensor
  - **node3**: resultant tensor

Add

In [1]:

```python
import tensorflow as tf

#Create nodes in computation graph
node1 = tf.constant(3, dtype=tf.int32)
node2 = tf.constant(5, dtype=tf.int32)
node3 = tf.add(node1, node2)

#Create session object
sess = tf.Session()
print("node1 + node2 = ", sess.run(node3))
#close the session
sess.close()
```

```
node1 + node2 =  8
```

- Quick Start Example

Add

```
import tensorflow as tf

#Create nodes in computation graph
node1 = tf.constant(3, dtype=tf.int32)
node2 = tf.constant(5, dtype=tf.int32)
node3 = tf.add(node1, node2)

#Create session object
sess = tf.Session()
print("node1 + node2 = ", sess.run(node3))
#close the session
sess.close()
```

```
node1 + node2 =  8
```

Add 2

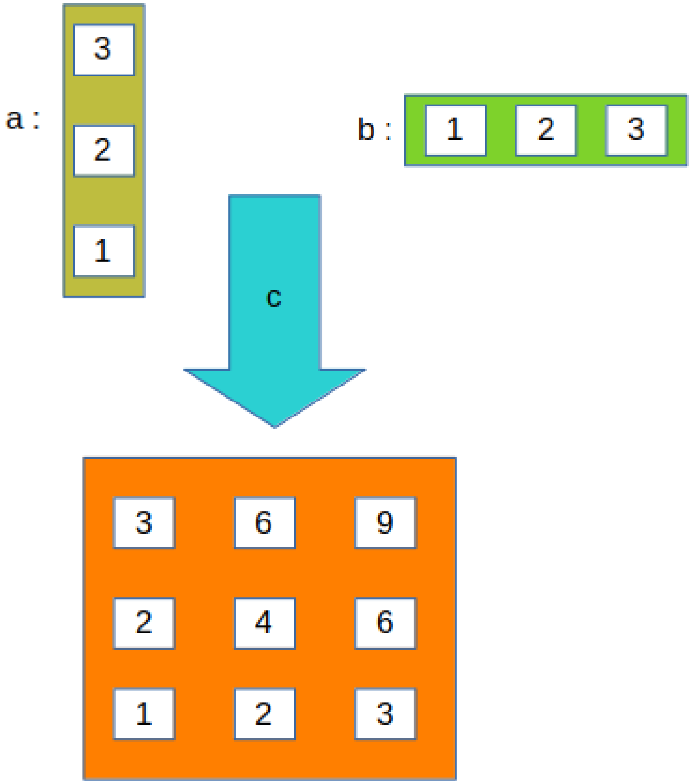
```
import tensorflow as tf

#Create nodes in computation graph
node1 = tf.constant(3, dtype=tf.int32)
node2 = tf.constant(5, dtype=tf.int32)
node3 = tf.add(node1, node2)

#Create session object
with tf.Session() as sess:
    print("node1 + node2 = ", sess.run(node3))
```

```
node1 + node2 =  8
```

- Example: Placeholder

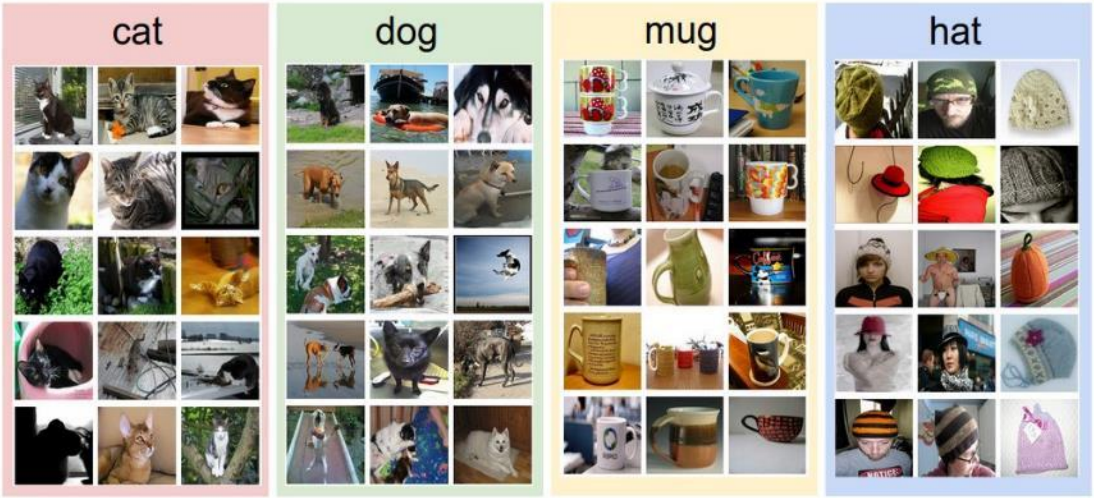
```
import tensorflow as tf

#Create nodes in computation graph
a = tf.placeholder(tf.int32, shape=(3,1))
b = tf.placeholder(tf.int32, shape=(1,3))
c = tf.matmul(a,b)

#Run computation graph
with tf.Session() as sess:
    print(sess.run(c, feed_dict={a:[[3],[2],[1]], b:[[1,2,3]]}))
```

```
[[3 6 9]
 [2 4 6]
 [1 2 3]]
```

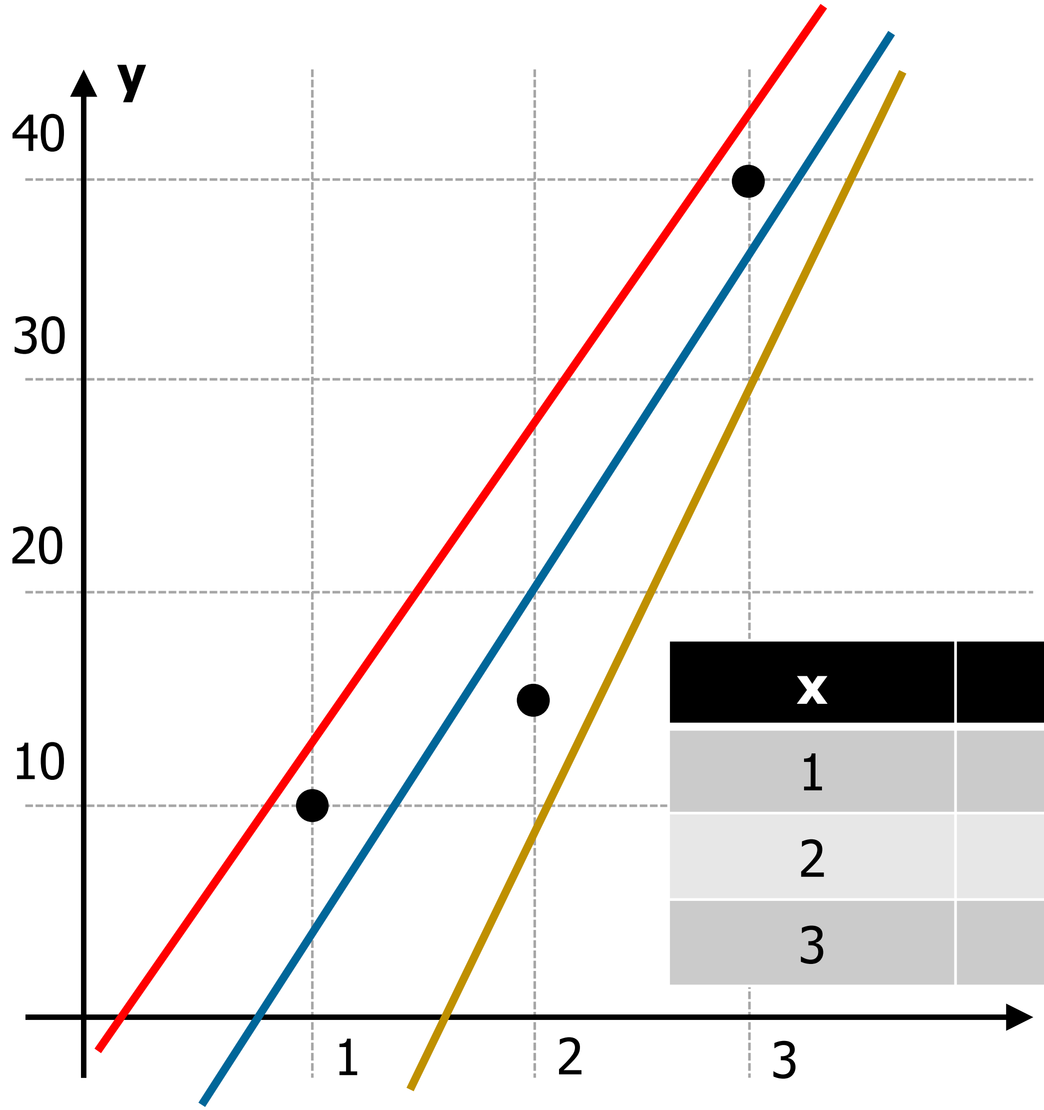


- Regression and Classification

Regression (Examples)	Classification (Examples)
• Exam Score Prediction (Linear Regression)	• Pass/Fail (Binary Classification) • Letter Grades (Multi-Level Classification)



- Linear model: $H(x) = Wx + b$
- Which model is the best among the given three?



x	y
1	10
2	15
3	40

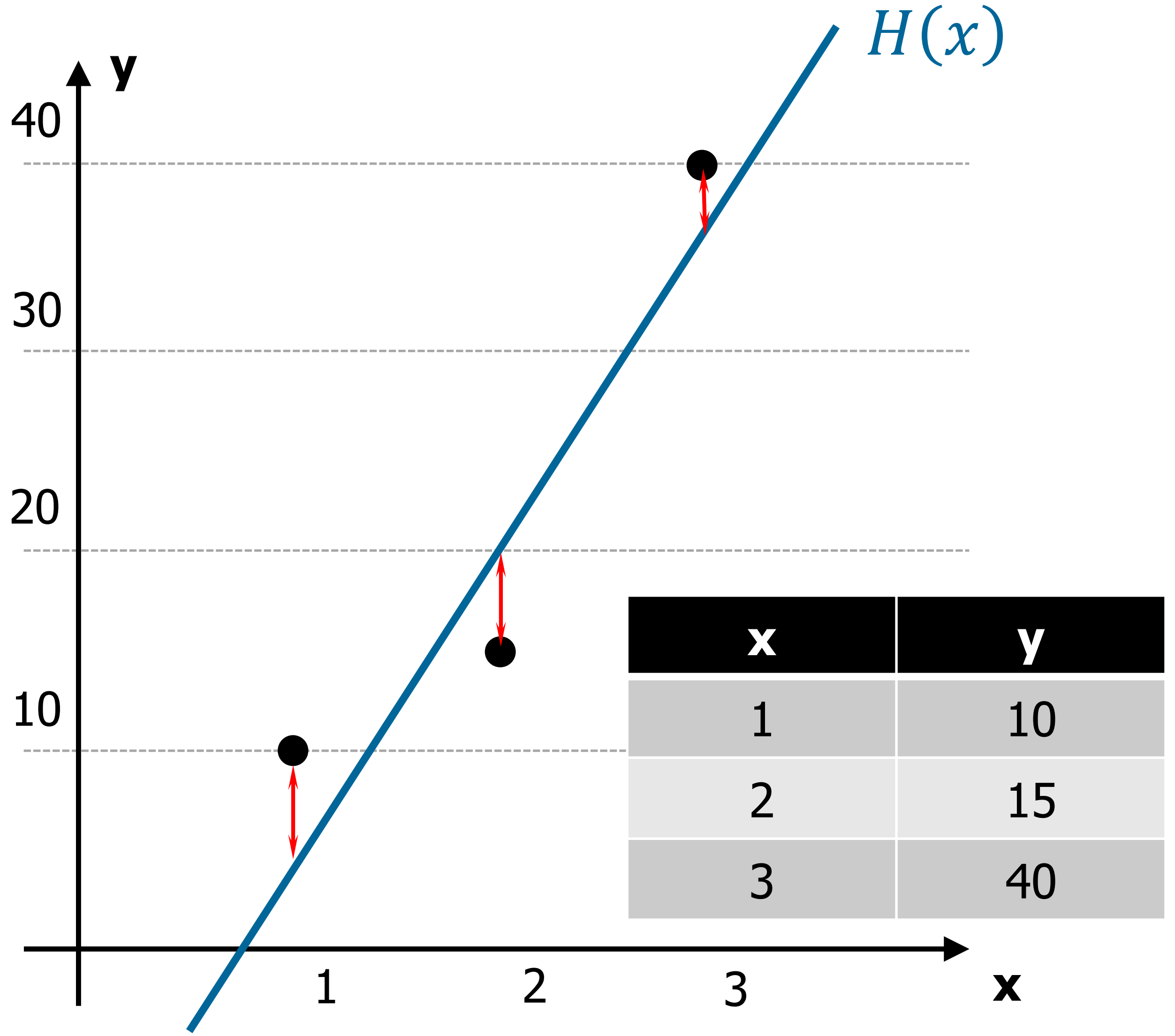
- Cost Function (or Loss Function)
  - How to fit the line to training data
  - The difference between model values and real measurements:

m: The number of training data

$$\frac{1}{m}\sum_{i=1}^{m}\left(H\left(x^i\right)-y^i\right)^2$$

$H(x) = Wx + b$

$$\text{Cost}(W,b) = \frac{1}{m}\sum_{i=1}^{m}\left(H\left(x^i\right)-y^i\right)^2$$

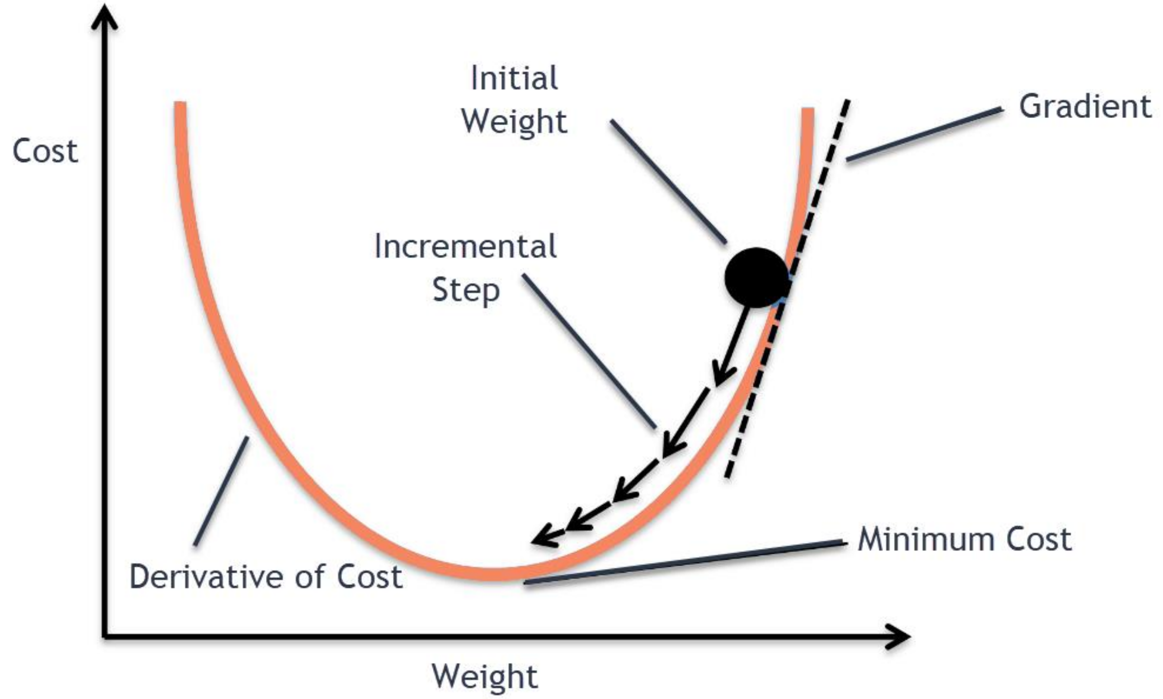


x	y
1	10
2	15
3	40

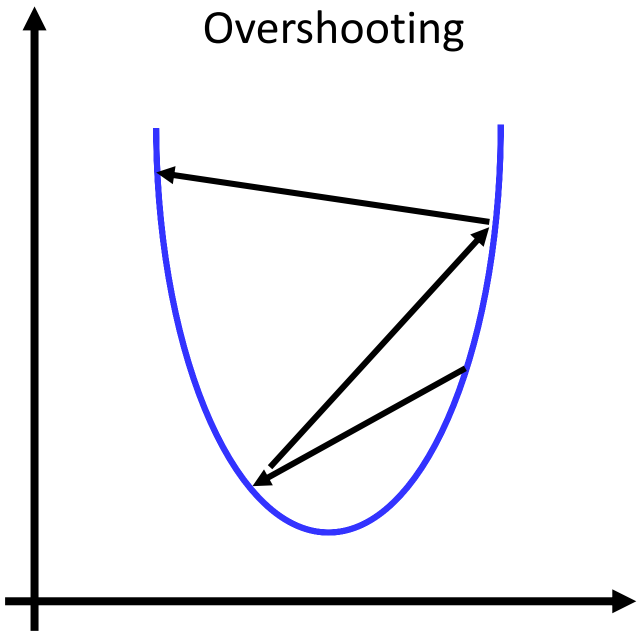
- Cost Function Minimization
  - Model: $H(x) = Wx + b$
  - Cost Function: $Cost(W,b) = \frac{1}{m}\sum_{i=1}^{m}\left(H\left(x^i\right) - y^i\right)^2 = \frac{1}{m}\sum_{i=1}^{m}\left(Wx^i + b - y^i\right)^2$
  - How to Minimize this Function? → **Gradient Descent Method**
    - Angle → Differentiation

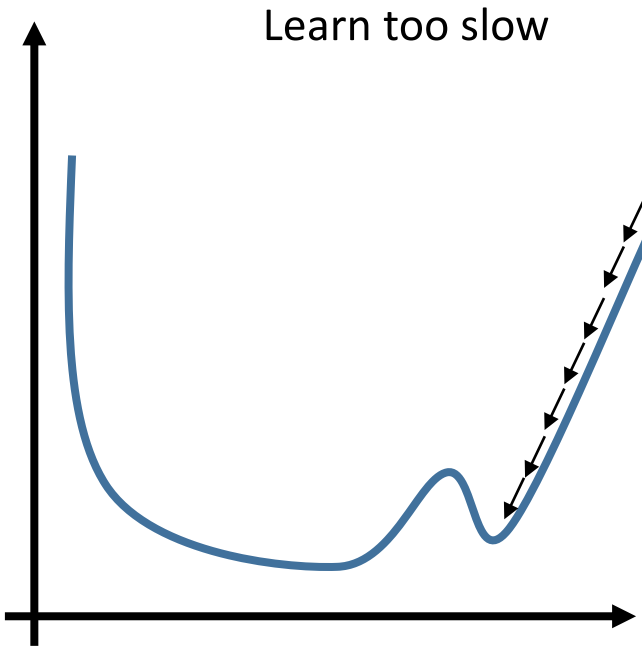
$$W \leftarrow W - \alpha \frac{\partial}{\partial W} Cost(W)$$

α: Learning rate

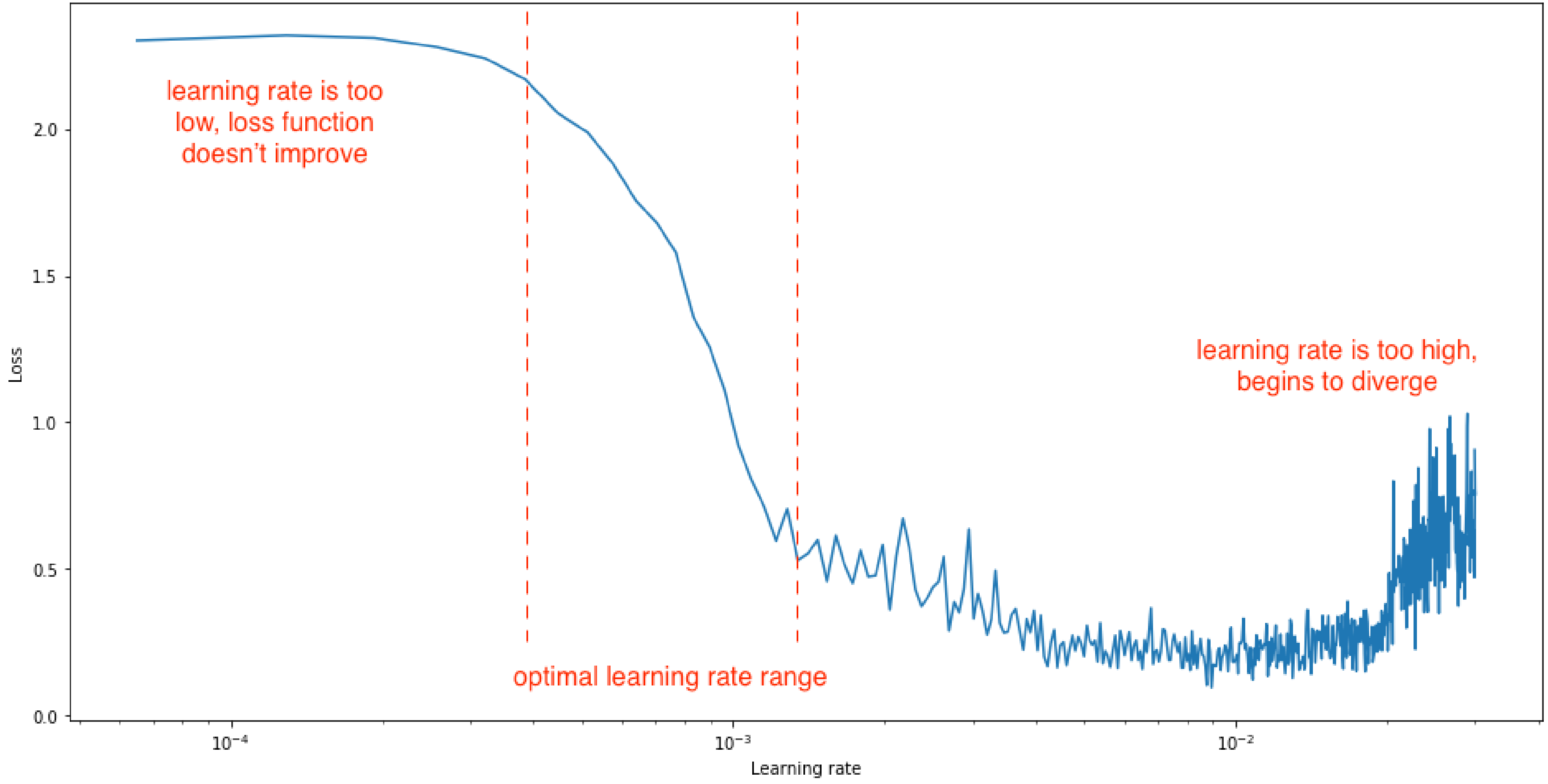


- Learning Rates
  - Too large: Overshooting
  - Too small: takes too long, stops in the middle
- How can we determine the learning rates?
  - Try several learning rates
    - Observe the cost function
    - Check it goes down in a reasonable rate



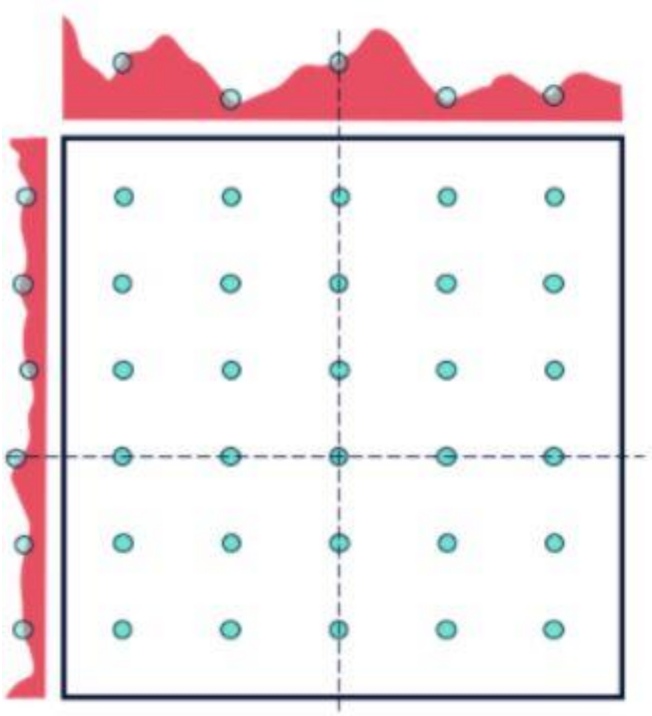


- How can we determine the learning rates?
  - Try several learning rates
    - Observe the cost function
    - Check it goes down in a reasonable rate

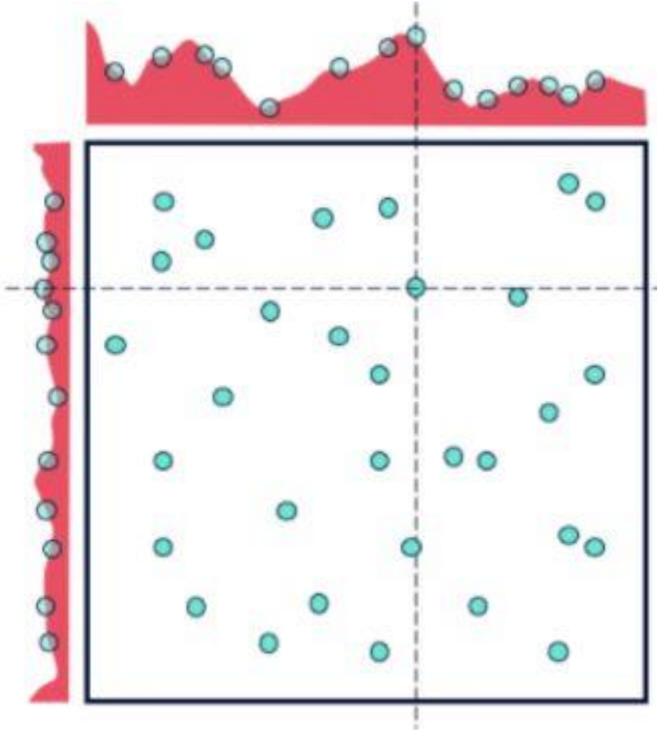


https://www.kaggle.com/residentmario/tuning-your-learning-rate

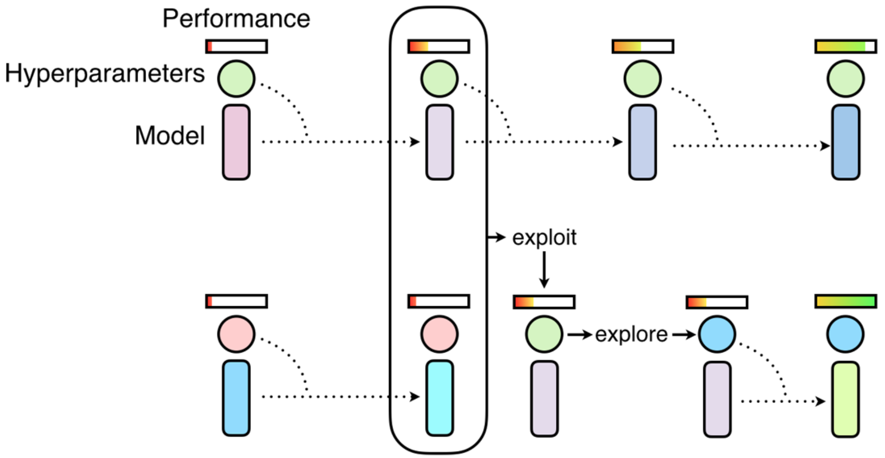
- How can we determine the learning rates?
  - Automating choice of learning rate
    - Grid Search
    - Random Search
    - Population Based Training



Grid Search

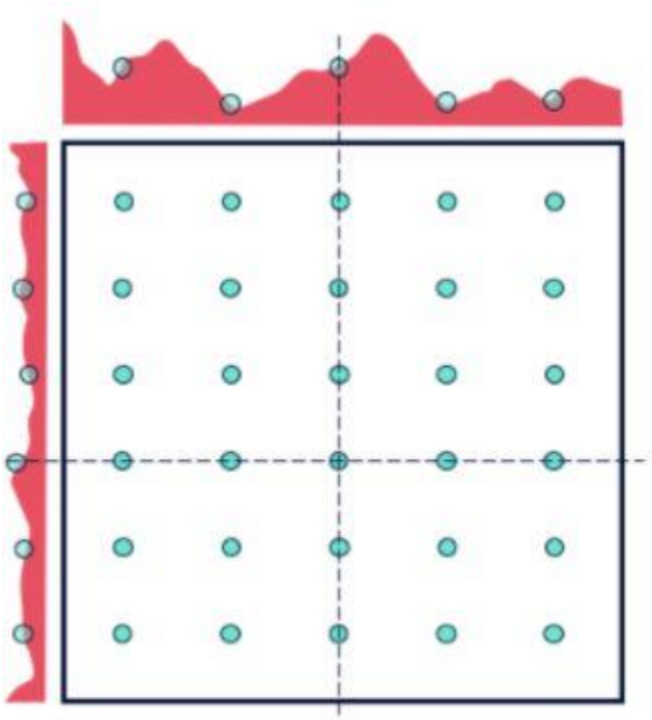


Random Search

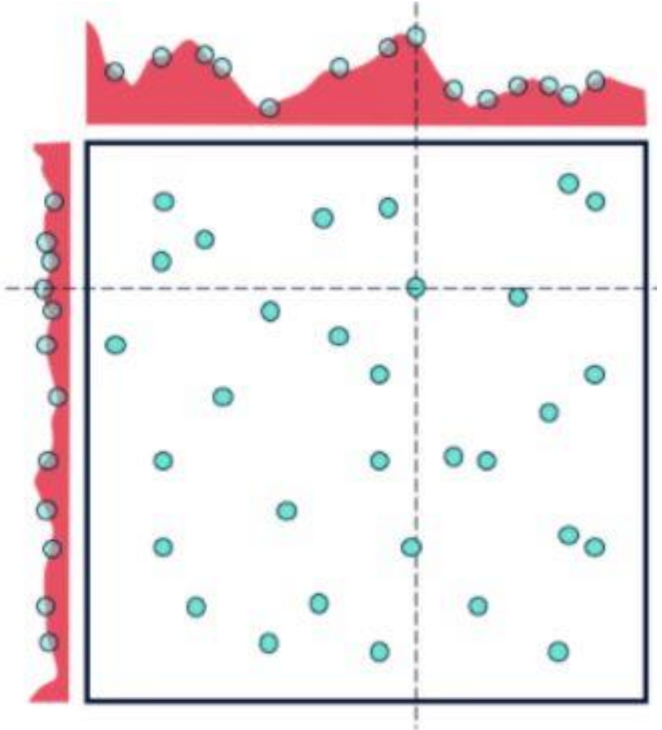


Population Based Training

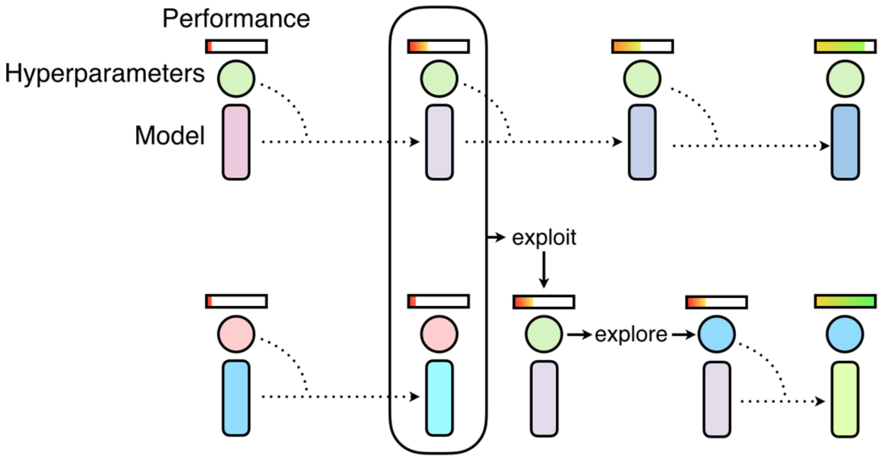
- How can we determine the learning rates?
  - Automating choice of learning rate
    - Grid Search
    - Random Search
    - Population Based Training



Grid Search



Random Search

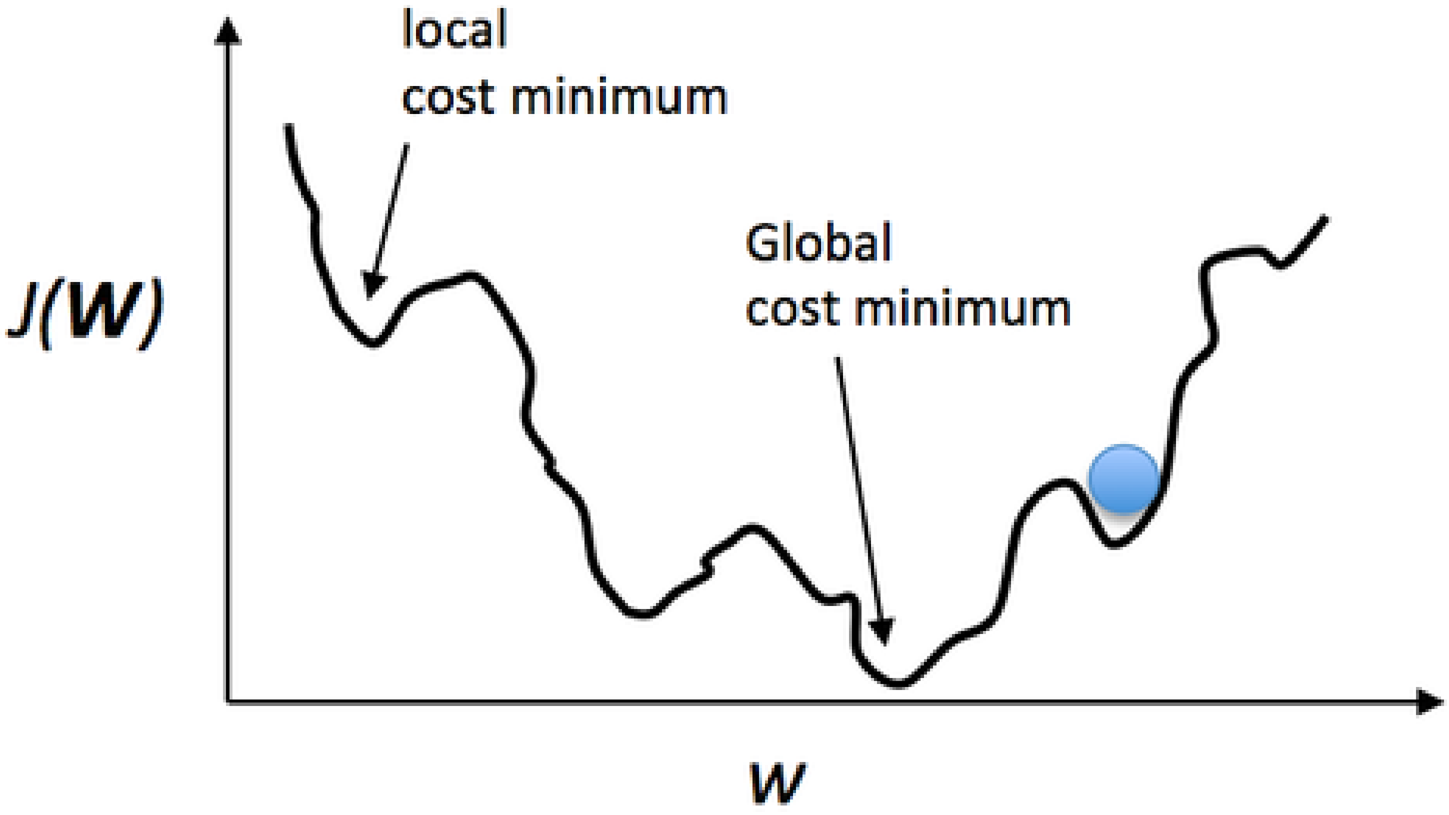


Population Based Training

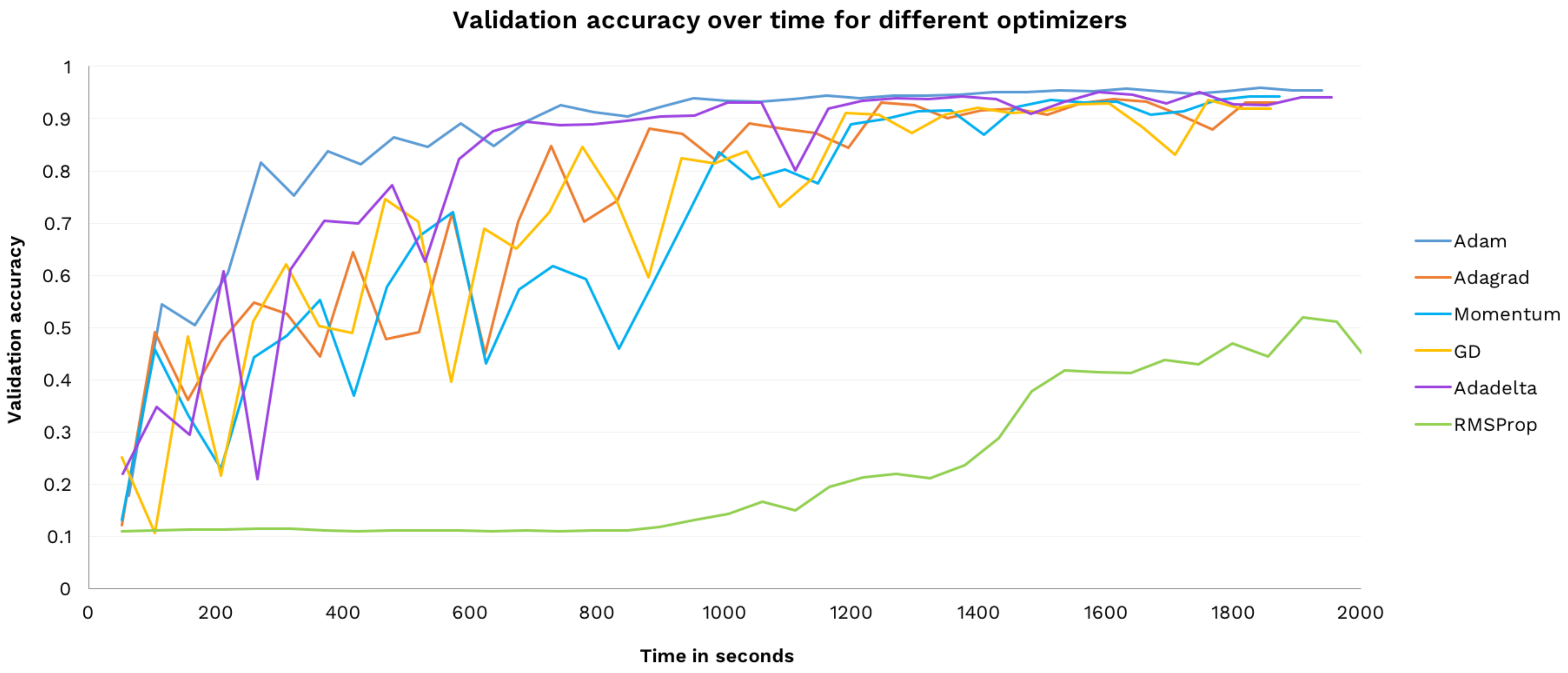
- Training with evolutionary competition

https://deepmind.com/blog/article/how-evolutionary-selection-can-train-more-capable-self-driving-cars

- Cost Function Minimization
  - Gradient Descent Method is only good for convex functions.



- Cost Function Minimization
  - Which optimizer performs best?



https://medium.com/octavian-ai/which-optimizer-and-learning-rate-should-i-use-for-deep-learning-5acb418f9b2

- Multi-Variable Linear Regression
  - Model:

$$H(x_1, x_2, \dots, x_n) = w_1 x_1 + w_2 x_2 + \cdots + w_n x_n + b$$

  - Cost:

$$Cost(W, b) = \frac{1}{m} \sum_{i=1}^{m} \left(H\left(x_1^i, x_2^i, \dots, x_n^i\right) - y^i \right)^2$$

KHU

- Multi-Variable Linear Regression
  - Model:

$$H(x_1, x_2, \dots, x_n) = w_1x_1 + w_2x_2 + \cdots + w_nx_n + b \Rightarrow H(X) = XW + b$$

$$\begin{pmatrix} x_1 & x_2 \dots & x_n \end{pmatrix} \cdot \begin{pmatrix} w_1 \\ w_2 \\ \dots \\ w_n \end{pmatrix} = w_1x_1 + w_2x_2 + \cdots + w_nx_n$$

X W

$$\Rightarrow H(X) = XW^T + b$$

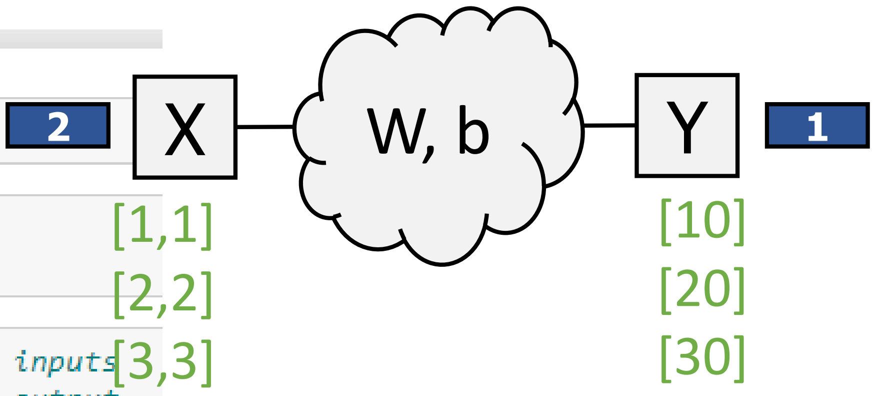
when $W = \begin{pmatrix} w_1 & w_2 \dots & w_n \end{pmatrix}$

Linear Regression Implementation (TensorFlow)

- TensorFlow
  - **Linear Regression**
- Keras
  - Linear Regression

Linear Regression Implementation (TensorFlow)

In [1]:
```python
import tensorflow as tf # Import the tensorflow lib
```



In [2]:
```python
x_data = [[1,1],[2,2],[3,3]] # Input data
y_data = [[10],[20],[30]] # Label data
```

In [3]:
```python
X = tf.placeholder(tf.float32, shape=[None,2]) #Construct the placeholder for inputs
Y = tf.placeholder(tf.float32, shape=[None,1]) #Construct the placeholder for output
```

In [4]:
```python
W=tf.Variable(tf.random_normal([2,1])) #Construct the Weight metrix
b=tf.Variable(tf.random_normal([1])) #Construct the bias vector
```

In [5]:
```python
model = tf.matmul(X,W)+b # Construct simple prediction model
cost = tf.reduce_mean(tf.square(model-Y)) #Define the loss function
train = tf.train.GradientDescentOptimizer(0.01).minimize(cost) # Define the optimizer
```

Model, Cost, Train

In [6]:
```python
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    #Training
    for step in range(2001):
        c, W_, b_, _ = sess.run([cost, W, b, train], feed_dict={X: x_data, Y: y_data})
        print(step, c, W_, b_)
    #Testing
    print("Testing",sess.run(model, feed_dict={X: [[4,4]]}))
```

Linear Regression Implementation (TensorFlow)

```
 [5.109964]] [0.01917489]
1992 5.258784e-05 [[4.881712]
 [5.109975]] [0.01912481]
1993 5.231946e-05 [[4.881723]
 [5.109986]] [0.01907484]
1994 5.2037543e-05 [[4.881734]
 [5.109997]] [0.01902499]
1995 5.1763218e-05 [[4.881745]
 [5.110008]] [0.01897525]
1996 5.1501138e-05 [[4.881756 ]
 [5.1100187]] [0.01892565]
1997 5.123555e-05 [[4.881767 ]
 [5.1100297]] [0.01887617]
1998 5.0959206e-05 [[4.8817773]
 [5.11004  ]] [0.01882678]
1999 5.0692397e-05 [[4.881788 ]
 [5.1100507]] [0.01877754]
2000 5.0433042e-05 [[4.8817983]
 [5.110061 ]] [0.01872844]
Testing [[39.986168]]
```

Open source Machine Learning Library

	Languages	Tutorials and training materials	CNN modeling capability	RNN modeling capability	Architecture: easy-to-use and modular front end	Speed	Multiple GPU support	Keras compatible
Theano	Python, C++	++	++	++	+	++	+	+
Tensor Flow	Python, C++, Java	+++	+++	++	+++	++	++	+
Torch	Lua	+	+++	++	++	+++	++	
Pytorch	Python	+	+++	++	++	+++	++	
Cafee	C++	+	++		+	+	+	
MXNet	R, Python, Julia, Scala	++	++	+	++	++	+++	
Neon	Python	+	++	+	+	++	+	
CNTK	C++	+	+	+++	+	++	+	

Modified from Source: https://www.svds.com/getting-started-deep-learning/

- TensorFlow is an open source software library for numerical computation using data flow graphs
- TensorFlow supports popular programming languages such as Python, C++, Java
- TensorFlow was originally developed by researchers and engineers working on the ***Google Brain Team*** within ***Google's Machine Intelligence research organization*** for the purposes of *conducting machine learning and deep neural networks research*, but the system is general enough to be applicable in a wide variety of other domains as well

Installation Procedures

Installing Anaconda on Window

- Installing Anaconda Environment on Window

Download the Anaconda 64-Bit Graphical Installer from the following link

https://www.anaconda.com/products/individual

Anaconda Installers

Windows

Python 3.8

64-Bit Graphical Installer (466 MB)

32-Bit Graphical Installer (397 MB)

MacOS

Python 3.8

64-Bit Graphical Installer (462 MB)

64-Bit Command Line Installer (454 MB)

Linux

Python 3.8

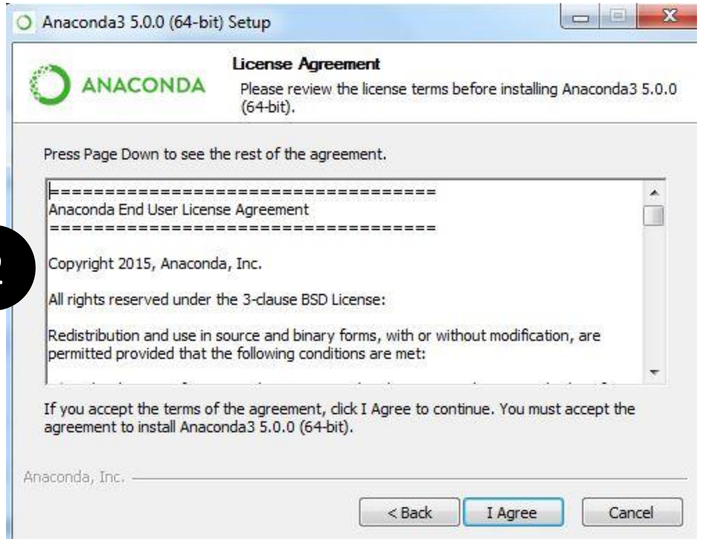
64-Bit (x86) Installer (550 MB)

64-Bit (Power8 and Power9) Installer (290 MB)

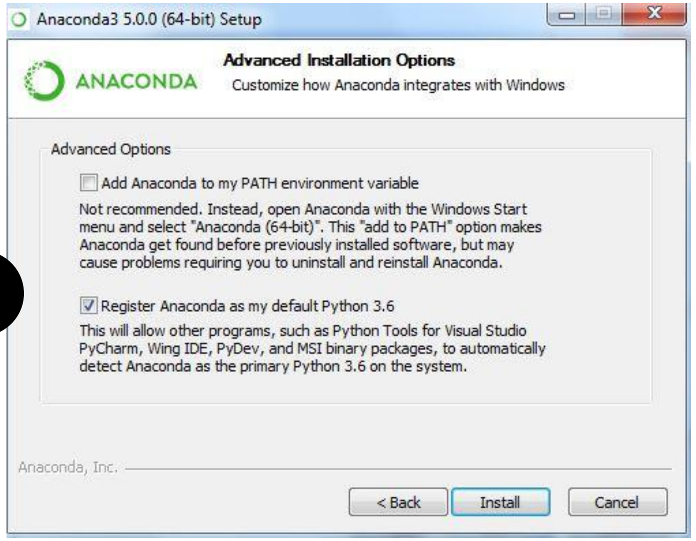
Installing Anaconda on Window

1

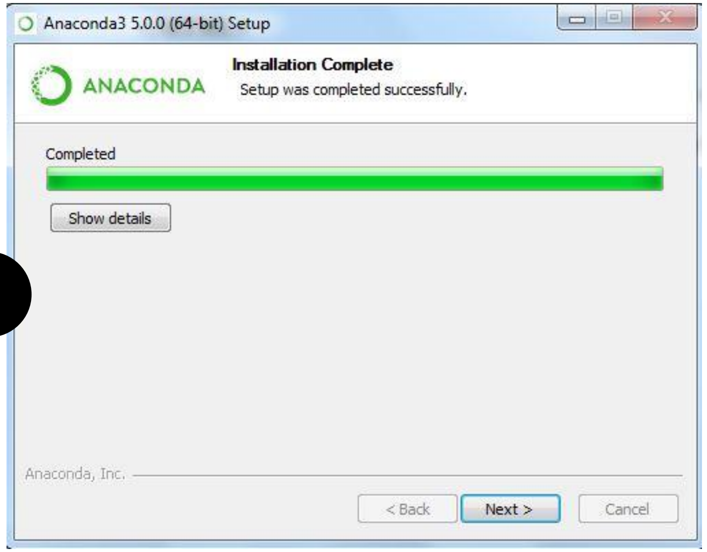
2



3

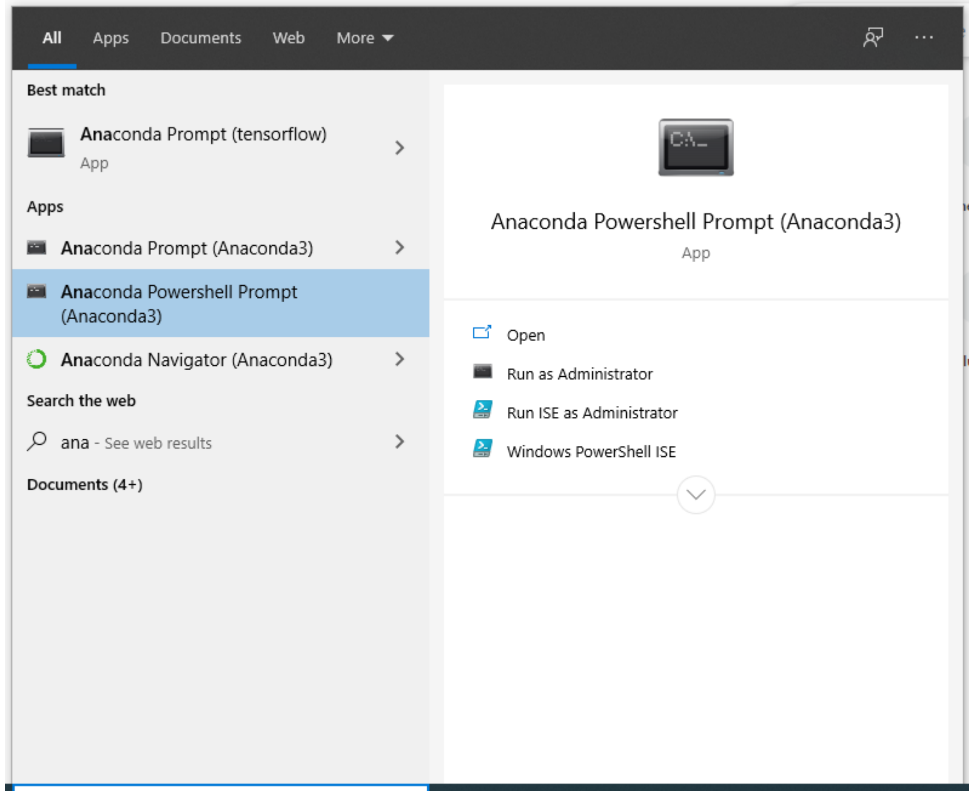


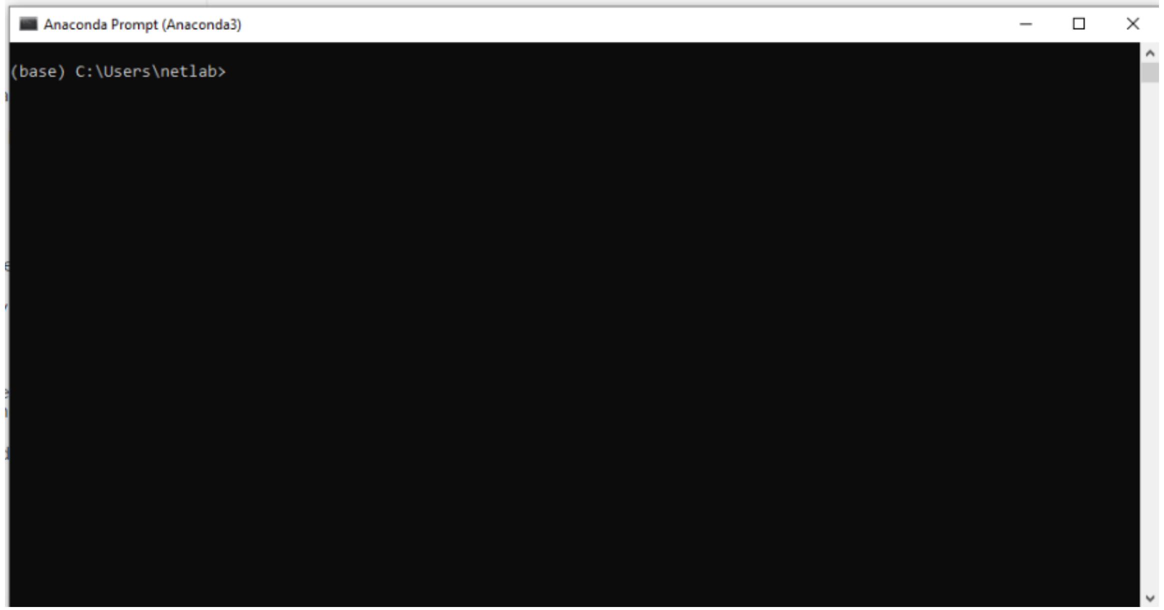
4



Constructing the Anaconda Virtual Environment

- After finishing Anaconda installation, open the “Anaconda Commend Prompt” to create the virtual environment





Start menu, search for and open “Anaconda Prompt”

Installing Tensorflow

- Type the following commands in the Anaconda Commend Prompt

Choose a name for your TensorFlow environment, such as “tf”.

```
conda create -n tf tensorflow==1.15
```

Activate the “tf” environment

```
conda activate tf
```

Anaconda Prompt (Anaconda3)

```
tensorflow-base-2.1. | 35.4 MB   | ########################################################################### | 100%
wheel-0.35.1         | 37 KB     | ########################################################################### | 100%
tensorflow-2.1.0     | 4 KB      | ########################################################################### | 100%
mkl-2020.2           | 109.3 MB  | ########################################################################### | 100%
intel-openmp-2020.2  | 1.6 MB    | ########################################################################### | 100%
cryptography-3.0     | 538 KB    | ########################################################################### | 100%
Preparing transaction: done
Verifying transaction: /
SafetyError: The package for chardet located at C:\Users\netlab\anaconda3\pkgs\chardet-3.0.4-py37_1003
appears to be corrupted. The path 'Lib/site-packages/chardet-3.0.4.dist-info/INSTALLER'
has an incorrect size.
  reported size: 5 bytes
  actual size: 4 bytes


done
Executing transaction: done
#
# To activate this environment, use
#
#     $ conda activate tf
#
# To deactivate an active environment, use
#
#     $ conda deactivate


(base) C:\Users\netlab>conda activate tf

(tf) C:\Users\netlab>_
```

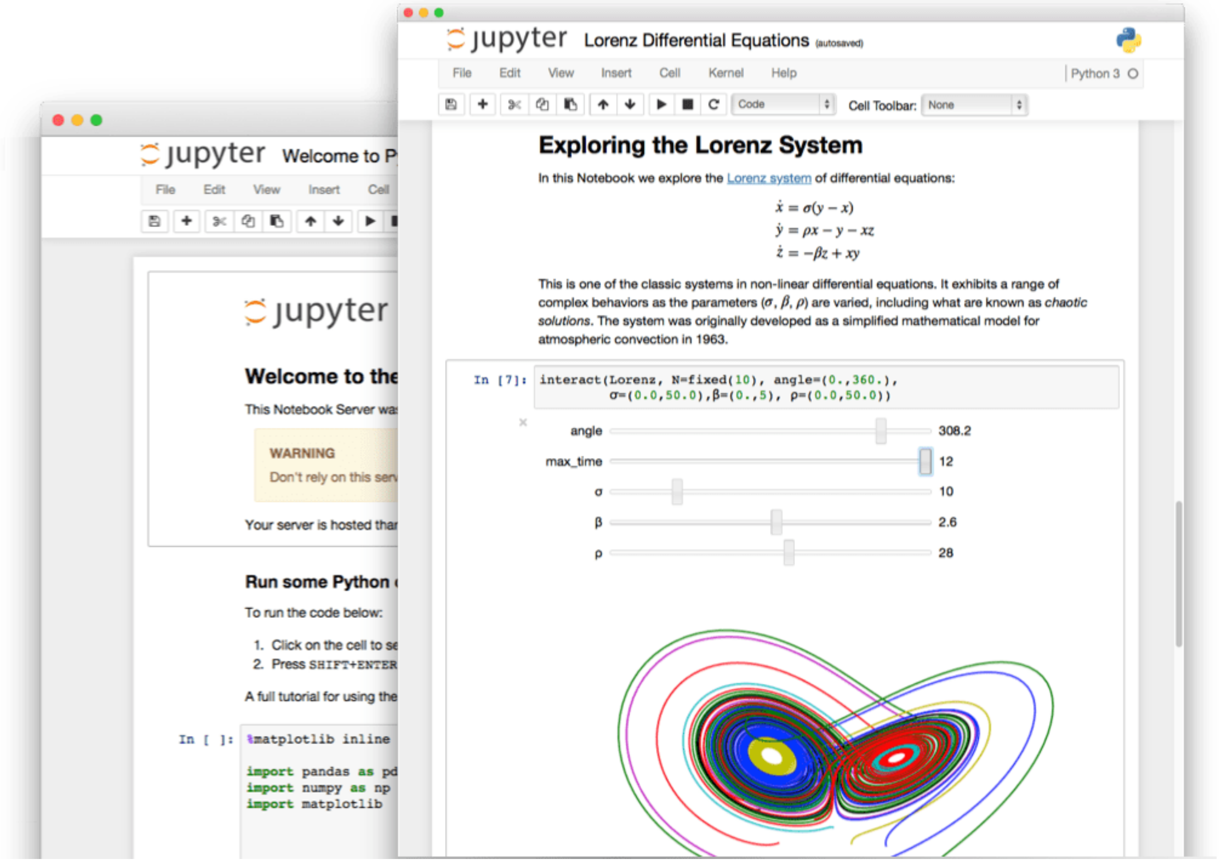
https://docs.anaconda.com/anaconda/user-guide/tasks/tensorflow/

Installing Jupyter Notebook with Anaconda

- Type the following commend in the Anaconda Commend Prompt (in the "tf" environment)

```
conda install jupyter
```

The Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and narrative text. Uses include: data cleaning and transformation, numerical simulation, statistical modeling, data visualization, machine learning, and much more.



https://jupyter.org/

Installing Keras

- Type the following commend in the Anaconda Commend Prompt (in the “tf” environment)

```
conda install keras
```

Keras

Deep learning for humans.
Keras is an API designed for human beings, not machines. Keras follows best practices for reducing cognitive load: it offers consistent & simple APIs, it minimizes the number of user actions required for common use cases, and it provides clear & actionable error messages. It also has extensive documentation and developer guides.

```
from tensorflow import keras
from tensorflow.keras import layers

# Instantiate a trained vision model
vision_model = keras.applications.ResNet50()

# This is our video.encoding branch using the trained vision_model
video_input = keras.Input(shape=(100, None, None, 3))
encoded_frame_sequence = layers.TimeDistributed(vision_model)(video_input)
encoded_video = layers.LSTM(256)(encoded_frame_sequence)

# This is our text-processing branch for the question input
question_input = keras.Input(shape=(100,), dtype='int32')
embedded_question = layers.Embedding(10000, 256)(question_input)
encoded_question = layers.LSTM(256)(embedded_question)

# And this is our video question answering model:
merged = keras.layers.concatenate([encoded_video, encoded_question])
output = keras.layers.Dense(1000, activation='softmax')(merged)
video_qa_model = keras.Model(inputs=[video_input, question_input],
                             outputs=output)
```

https://keras.io/

Installing Numpy

- Type the following commend in the Anaconda Commend Prompt (in the "tf" environment)

```
conda install numpy
```

NumPy

POWERFUL N-DIMENSIONAL ARRAYS

Fast and versatile, the NumPy vectorization, indexing, and broadcasting concepts are the de-facto standards of array computing today.

NUMERICAL COMPUTING TOOLS

NumPy offers comprehensive mathematical functions, random number generators, linear algebra routines, Fourier transforms, and more.

INTEROPERABLE

NumPy supports a wide range of hardware and computing platforms, and plays well with distributed, GPU, and sparse array libraries.

PERFORMANT

The core of NumPy is well-optimized C code. Enjoy the flexibility of Python with the speed of compiled code.

EASY TO USE

NumPy's high level syntax makes it accessible and productive for programmers from any background or experience level.

OPEN SOURCE

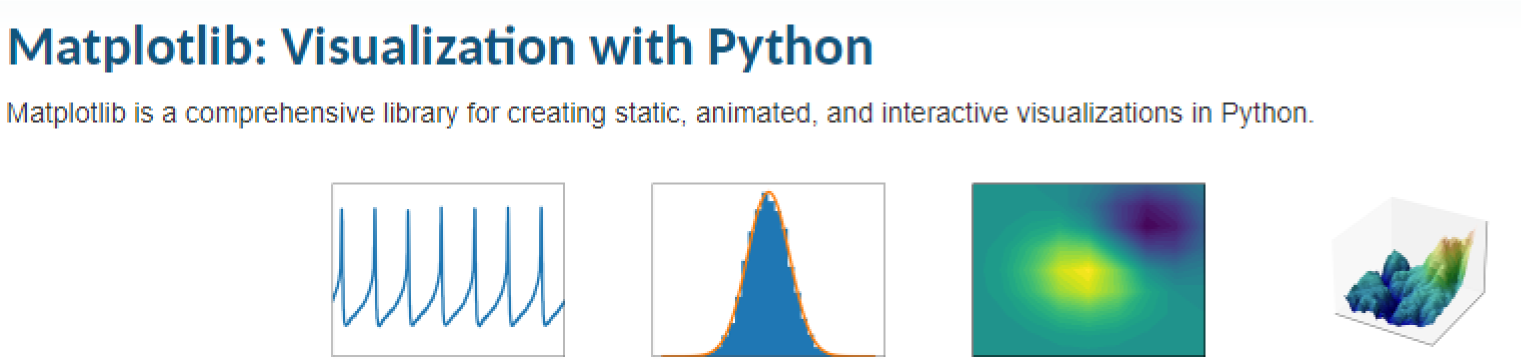
Distributed under a liberal BSD license, NumPy is developed and maintained publicly on GitHub by a vibrant, responsive, and diverse community.

https://numpy.org/pai

Installing Matplotlib

- Type the following commend in the Anaconda Commend Prompt (in the "tf" environment)

```
conda install matplotlib
```



https://matplotlib.org/

Installing scikit-learn

- Type the following commend in the Anaconda Commend Prompt (in the “tf” environment)

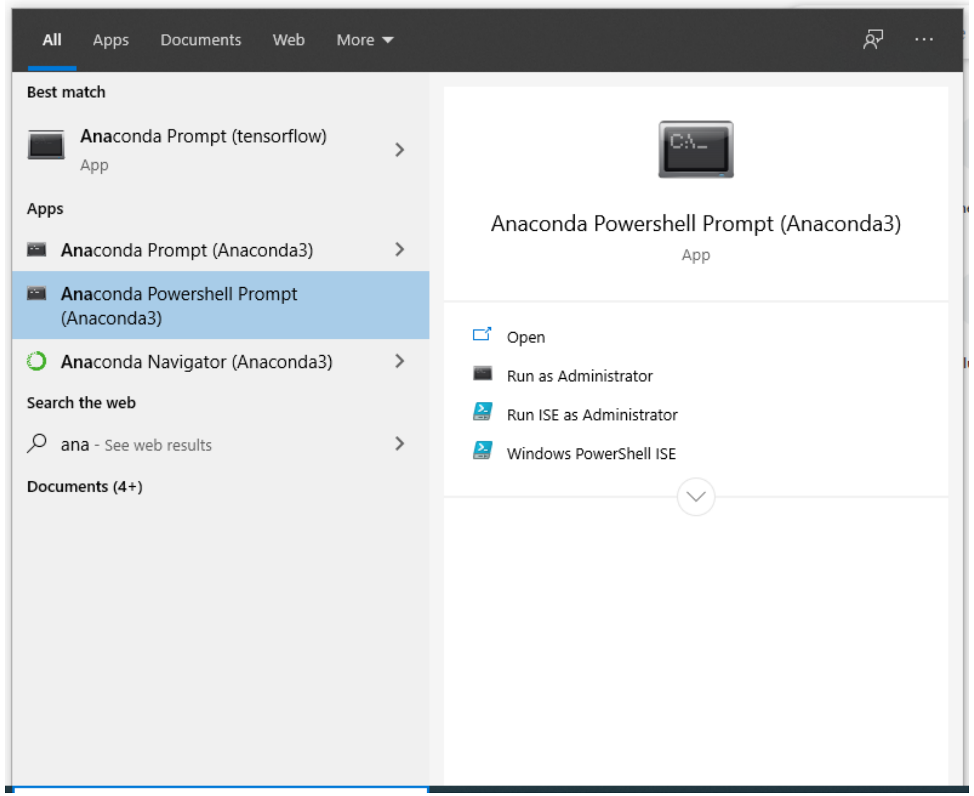
```
conda install scikit-learn
```

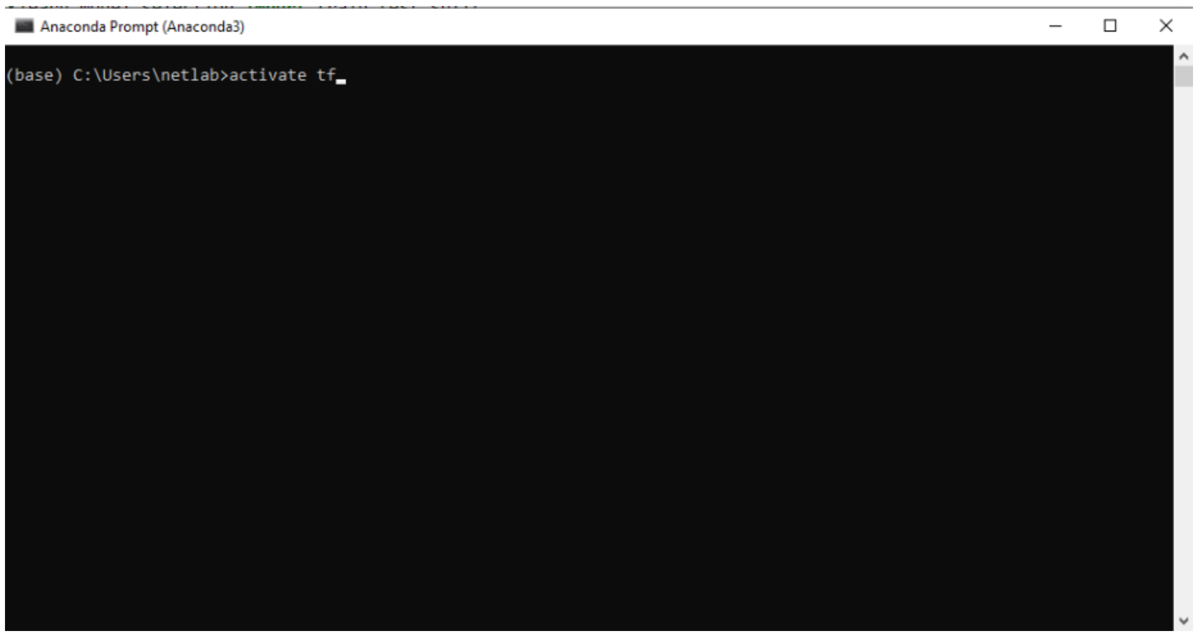
https://scikit-learn.org/stable/index.html

Opening Jupyter Notebook

Opening Jupyter Notebook

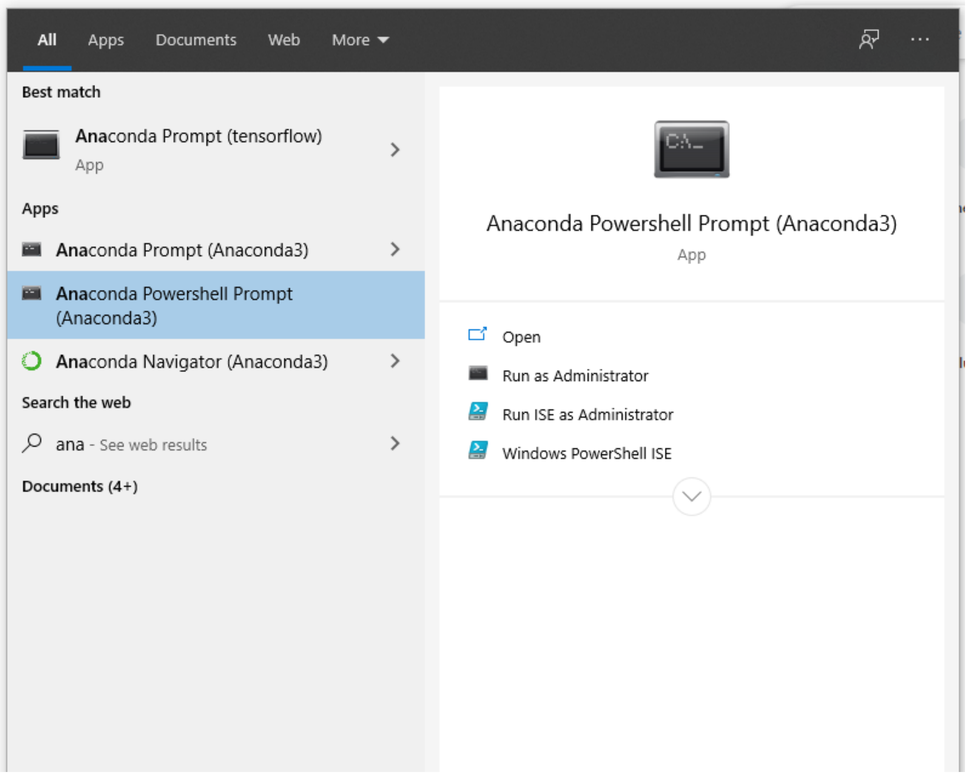
- Start menu, search for and open “Anaconda Prompt”
- Activate the “tf” environment

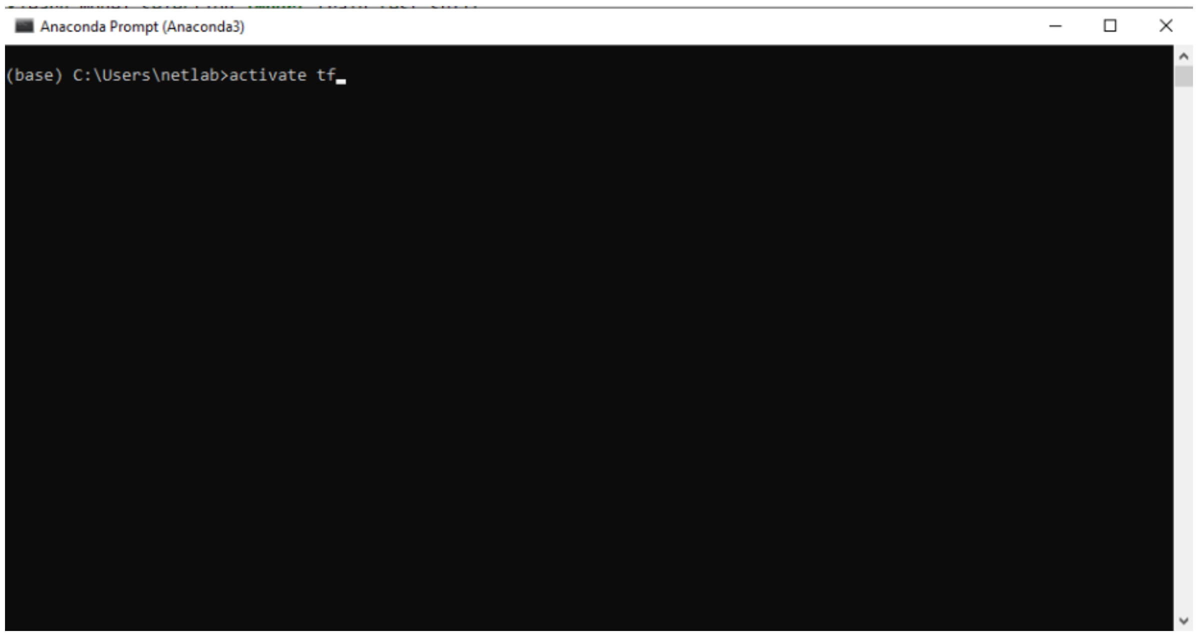




Opening Jupyter Notebook

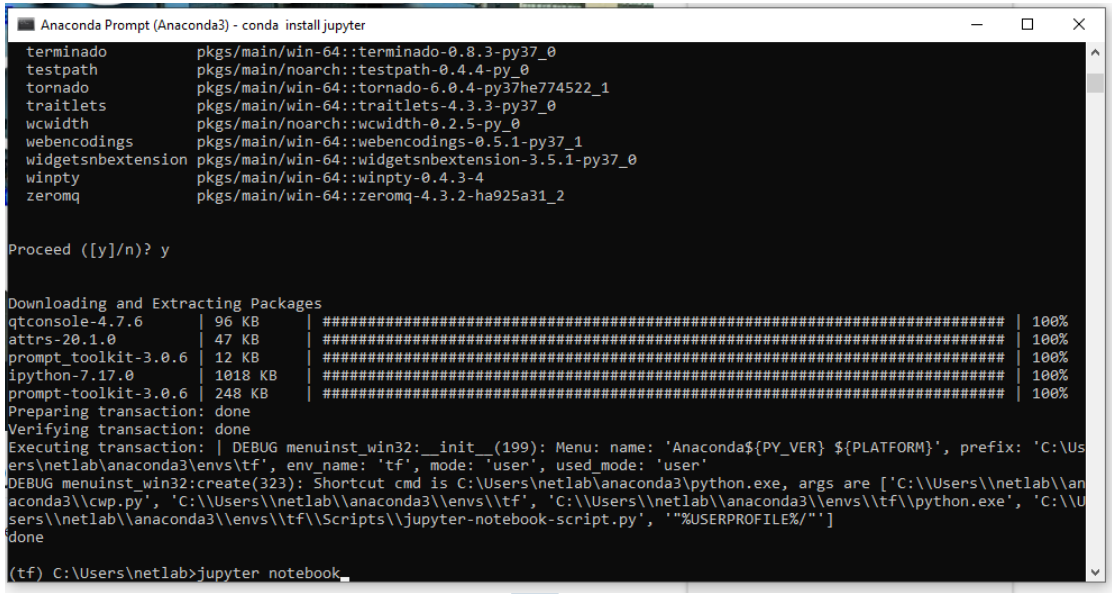
- Start menu, search for and open "Anaconda Prompt"
- Activate the "tf" environment

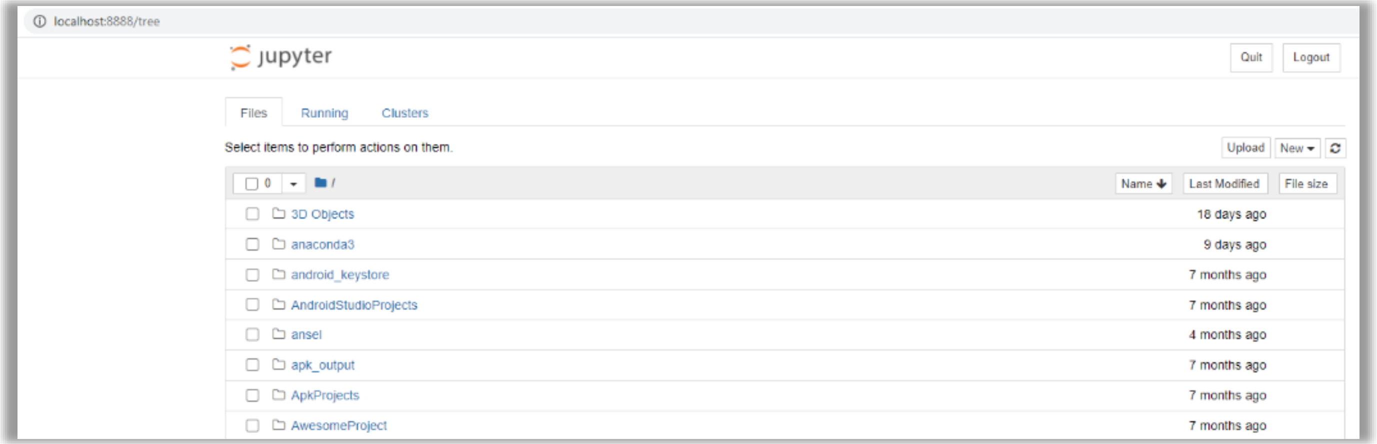




Opening Jupyter Notebook

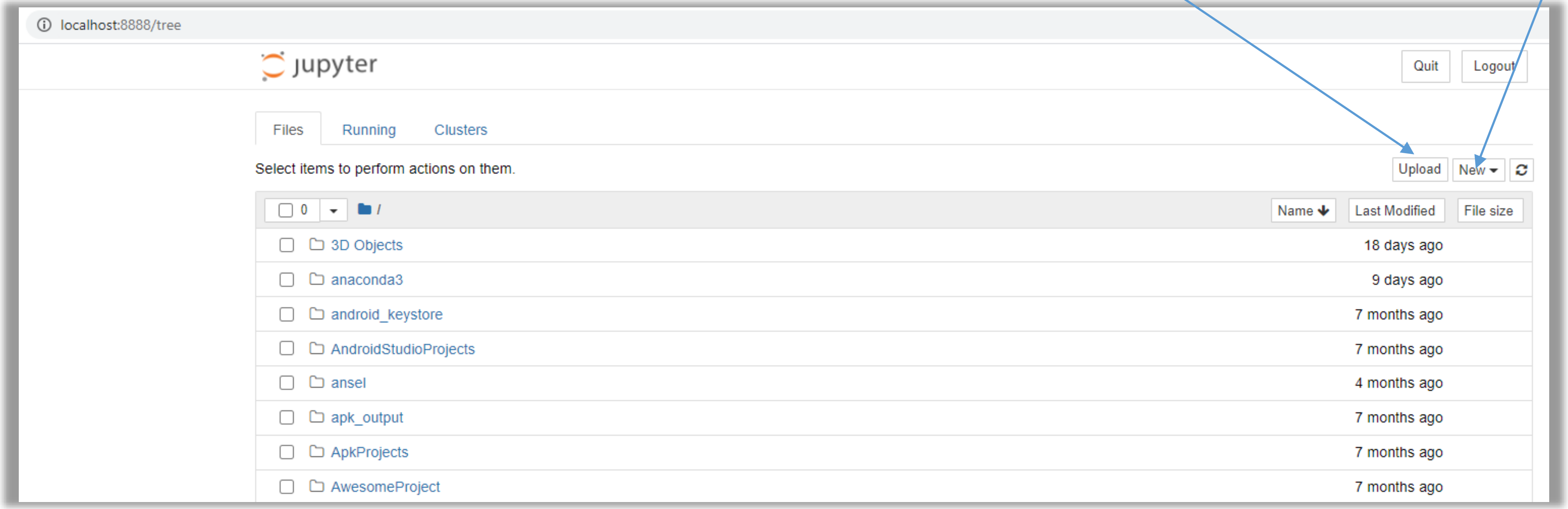
- Type jupyter notebook



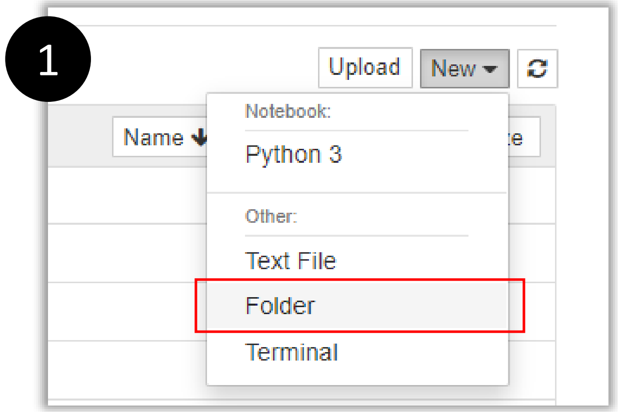


To upload files

Create new folders and files

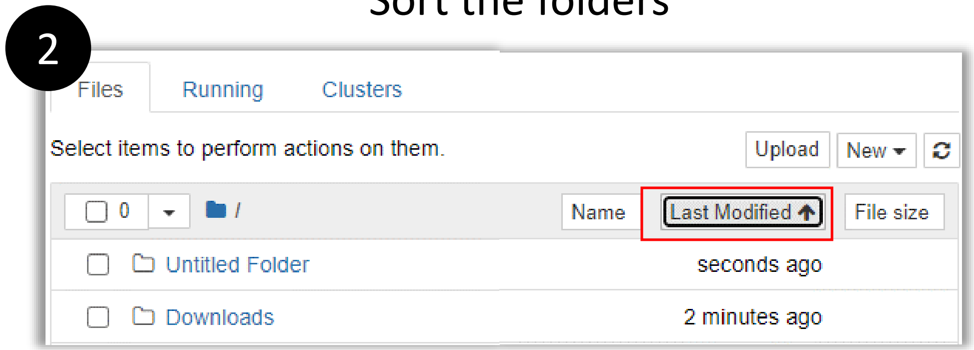


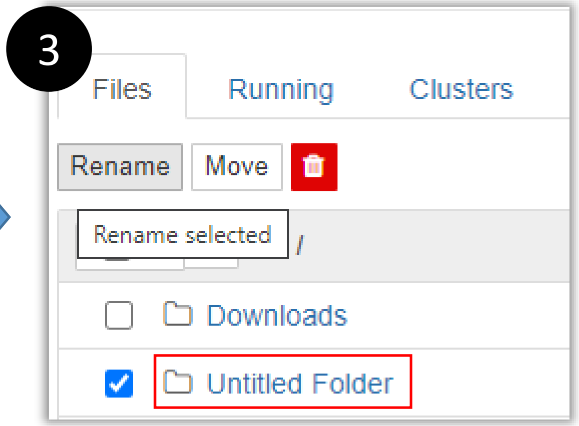
Creating new folder and file in Jupyter



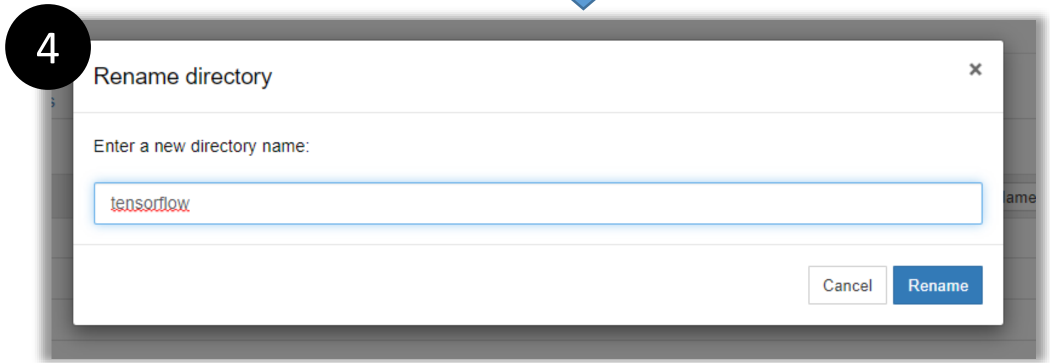
Create new folder

Sort the folders

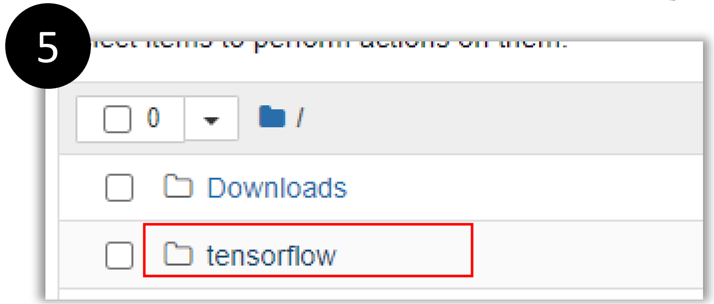




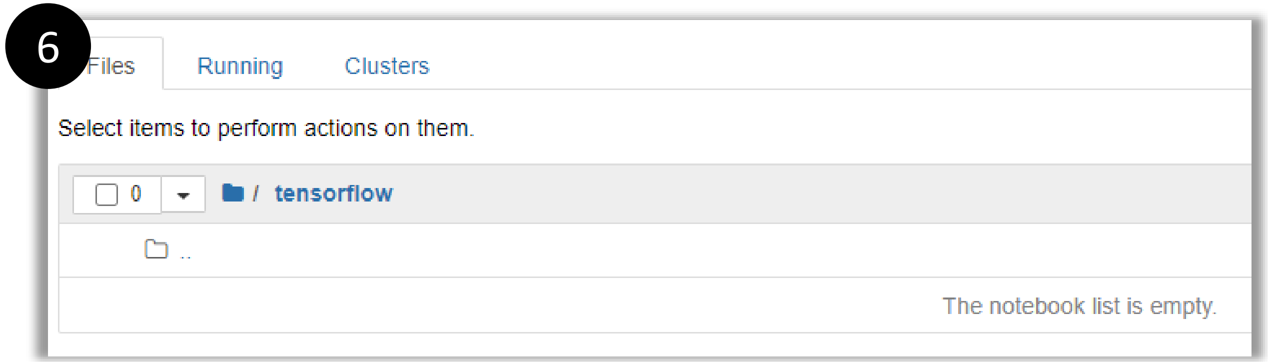
Click “Untitled Folder”
Click “Rename”



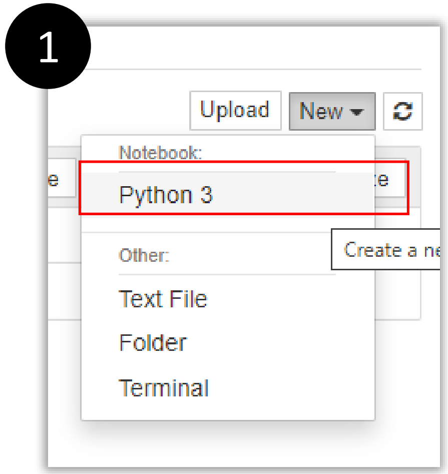
Rename the folder name as “tensorflow”



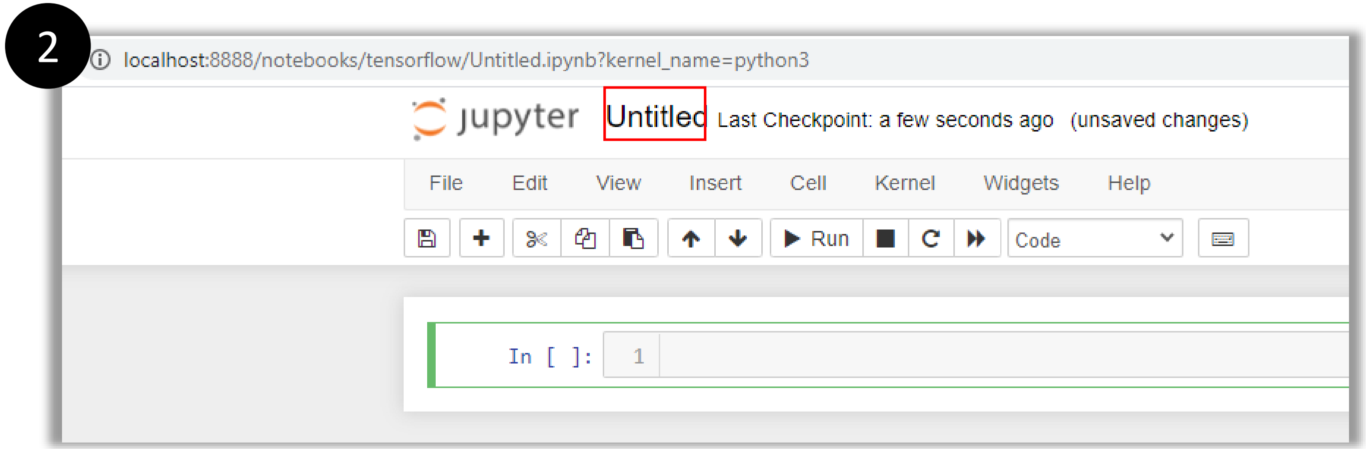
Enter into the “tensorflow” folder



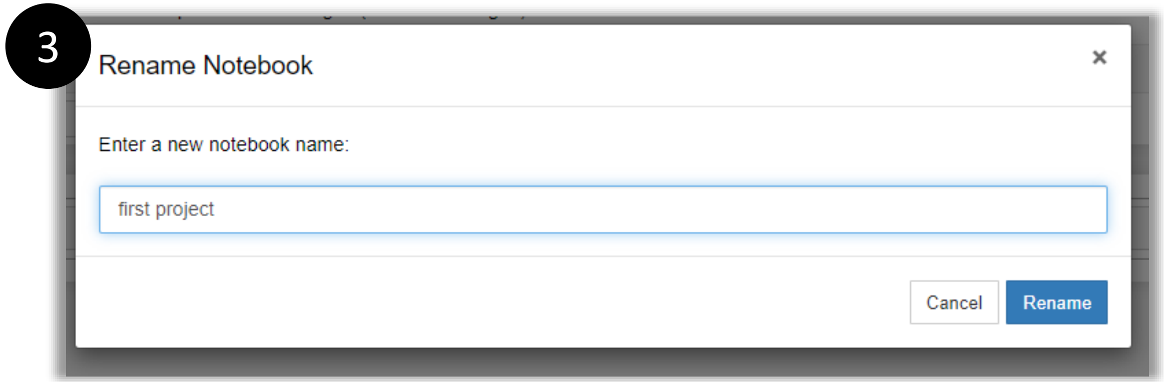
Creating new folder and file in Jupyter



Create new python 3 file

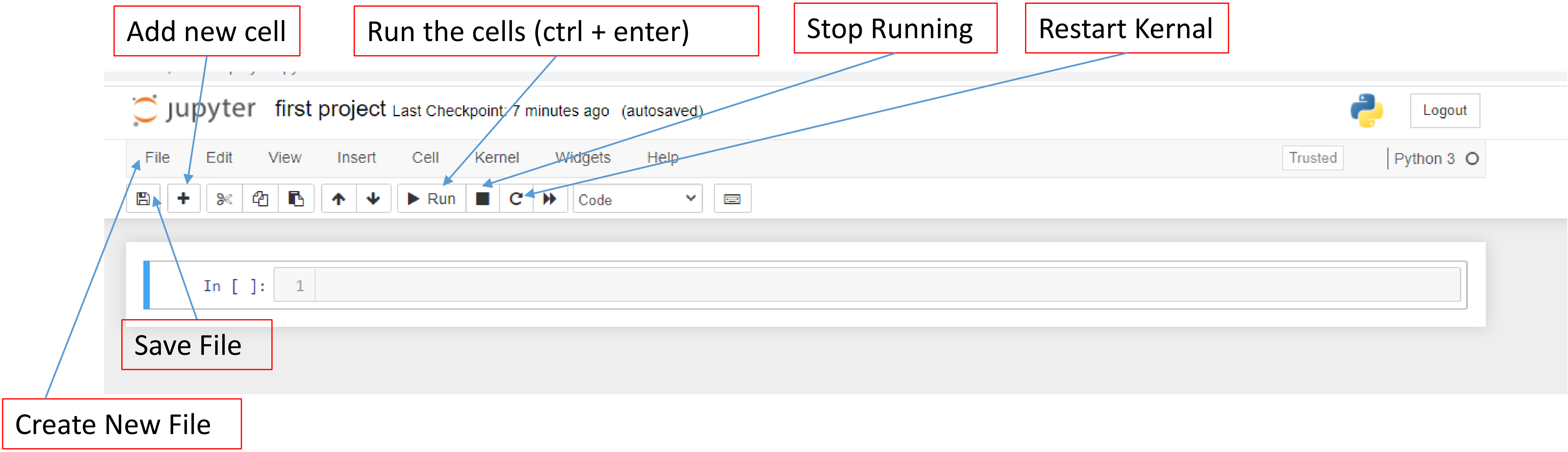


“Untitled” file will automatically appear and then rename the file name "Untitled”.



Rename the file name “Untitled” into “first project”

Jupyter Notebook project file



First Tensorflow Project

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Run Code

In [1]:
```python
import tensorflow as tf # Import the tensorflow lib
```

In [2]:
```python
x_data = [[1,1],[2,2],[3,3]] # Input data
y_data = [[10],[20],[30]] # Label data
```

In [3]:
```python
X = tf.placeholder(tf.float32, shape=[None,2]) #Construct the placeholder for inputs
Y = tf.placeholder(tf.float32, shape=[None,1]) #Construct the placeholder for output
```

In [4]:
```python
W=tf.Variable(tf.random_normal([2,1])) #Construct the Weight metrix
b=tf.Variable(tf.random_normal([1])) #Construct the bias vector
```

In [5]:
```python
model = tf.matmul(X,W)+b # Construct simple prediction model
cost = tf.reduce_mean(tf.square(model-Y)) #Define the loss function
train = tf.train.GradientDescentOptimizer(0.01).minimize(cost) # Define the optimizer
```

In [6]:
```python
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    #Training
    for step in range(2001):
        c, W_, b_, _ = sess.run([cost, W, b, train], feed_dict={X: x_data, Y: y_data})
        print(step, c, W_, b_)
    #Testing
    print("Testing",sess.run(model, feed_dict={X: [[4,4]]}))
```

First Tensorflow Project


```
train = tf.train.GradientDescentOptimizer(0.01).minimize(cost) # Define the optimizer
```

In [6]:

```python
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    #Training
    for step in range(2001):
        c, W_, b_, _ = sess.run([cost, W, b, train], feed_dict={X: x_data, Y: y_data})
        print(step, c, W_, b_)
    #Testing
    print("Testing",sess.run(model, feed_dict={X: [[4,4]]}))
```

```
 [5.109964]] [0.01917489]
1992 5.258784e-05 [[4.881712]
 [5.109975]] [0.01912481]
1993 5.231946e-05 [[4.881723]
 [5.109986]] [0.01907484]
1994 5.2037543e-05 [[4.881734]
 [5.109997]] [0.01902499]
1995 5.1763218e-05 [[4.881745]
 [5.110008]] [0.01897525]
1996 5.1501138e-05 [[4.881756 ]
 [5.1100187]] [0.01892565]
1997 5.123555e-05 [[4.881767 ]
 [5.1100297]] [0.01887617]
1998 5.0959206e-05 [[4.8817773]
 [5.11004  ]] [0.01882678]
1999 5.0692397e-05 [[4.881788 ]
 [5.1100507]] [0.01877754]
2000 5.0433042e-05 [[4.8817983]
 [5.110061 ]] [0.01872844]
Testing [[39.986168]]
```

KHU