



A Quick Introduction to Python and Machine Learning

Ling Zhang

Apr 17th, 2019

Objective

- Help on getting started with Python
- Introduce the basic concepts of neural network and how it is trained
- Help on getting started with machine learning programming using PyTorch (or Tensorflow)

Part I – Introduction to Python

- What is Python
- Commonly-used Python editors
- Data Structures in Python
- Control Structures (Selection and Loop)
- Python Functions
- Python Classes
- An Python package example: skrf

There are many Python tutorials available online. Here is an example:

<https://www.w3schools.com/python/default.asp>

Part I – Introduction to Python

- What is Python?
- Commonly-used Python editors
- Data Structures in Python
- Control Structures in Python
- Python Functions
- Python Classes
- An Python package example: skrf

What is Python

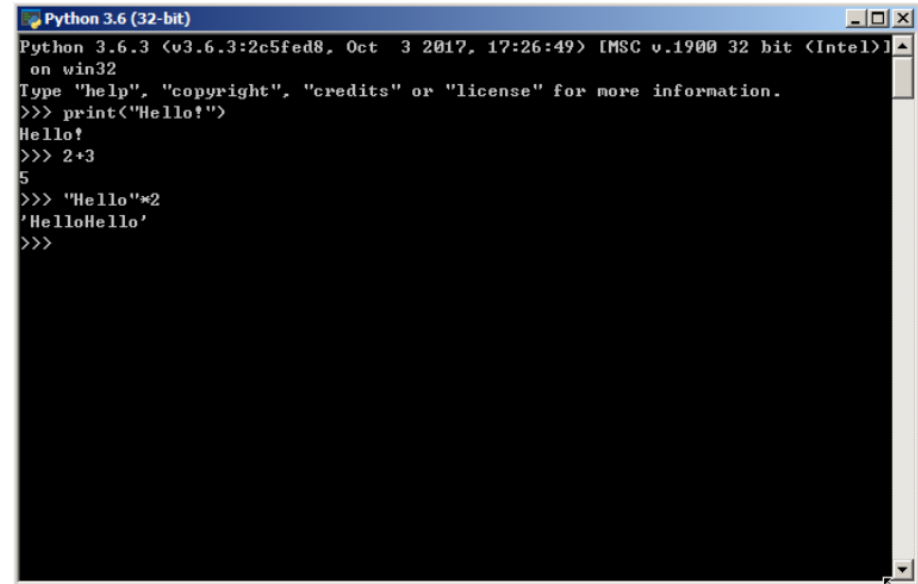
- Python is a general purpose programming language with the following nice features:
 - An object-based high-level programming language
 - A free software released under an open-source license
 - Cross-platform: Windows, Linux/Unix, MacOS X etc
 - A very readable and concise language with clear non-verbose syntax
 - A large variety of high-quality packages are available for various applications online and free to use
 - **Don't reinvent the wheel!**

Part I – Introduction to Python

- What is Python
- **Commonly-used Python editors**
- Data Structures in Python
- Control Structures (Selection and Loop)
- Python Functions
- Python Classes
- An Python package example: skrf

Python Terminal

- You can use the Python terminal for interactive computing in Python
- A preferred way is to use other editors like Jupyter Notebook or PyCharm

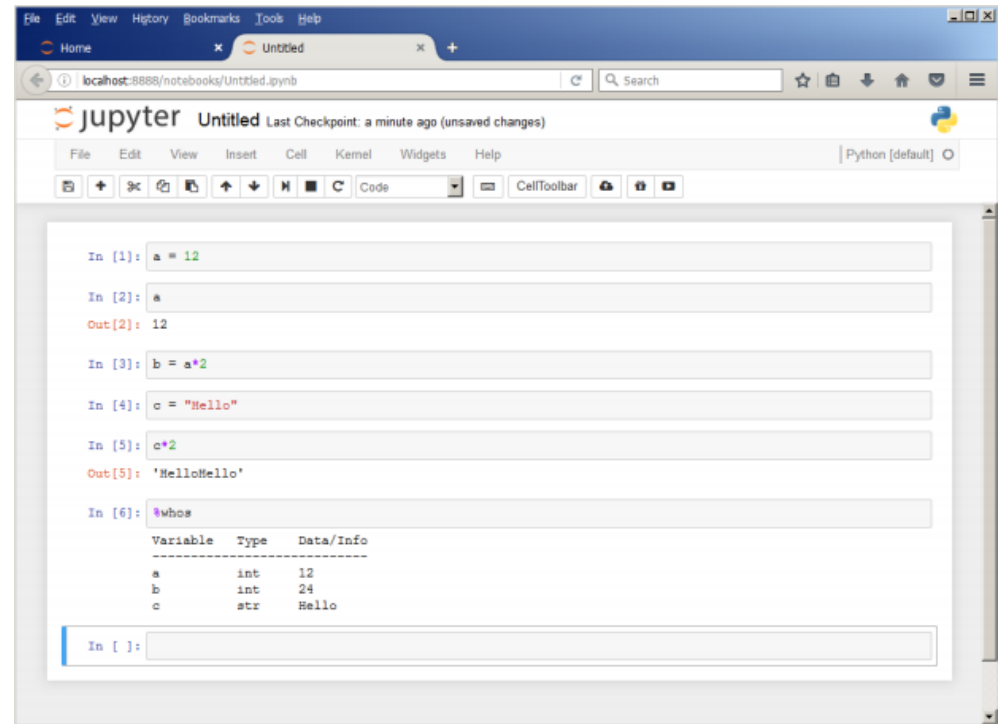


```
Python 3.6 (32-bit)
Python 3.6.3 (v3.6.3:2c5fed8, Oct 3 2017, 17:26:49) [MSC v.1900 32 bit (Intel)]
on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Hello!")
Hello!
>>> 2+3
5
>>> 'Hello'*2
'HelloHello'
>>>
```

Jupyter Notebook

<https://jupyter.org/>

- A web-based interactive computational environment
- Can contain code, text, mathematics, plots and rich media.
- Can work interactively
- Easy to visualize results
- Easy to share with others



The screenshot displays a Jupyter Notebook interface in a web browser. The browser's address bar shows the URL `localhost:8888/notebooks/Untitled.ipynb`. The notebook's title bar indicates the file is "Untitled" and shows the last checkpoint was "a minute ago (unsaved changes)". The interface includes a menu bar with options like File, Edit, View, Insert, Cell, Kernel, Widgets, and Help. Below the menu is a toolbar with various icons for cell operations. The main workspace contains six code cells:

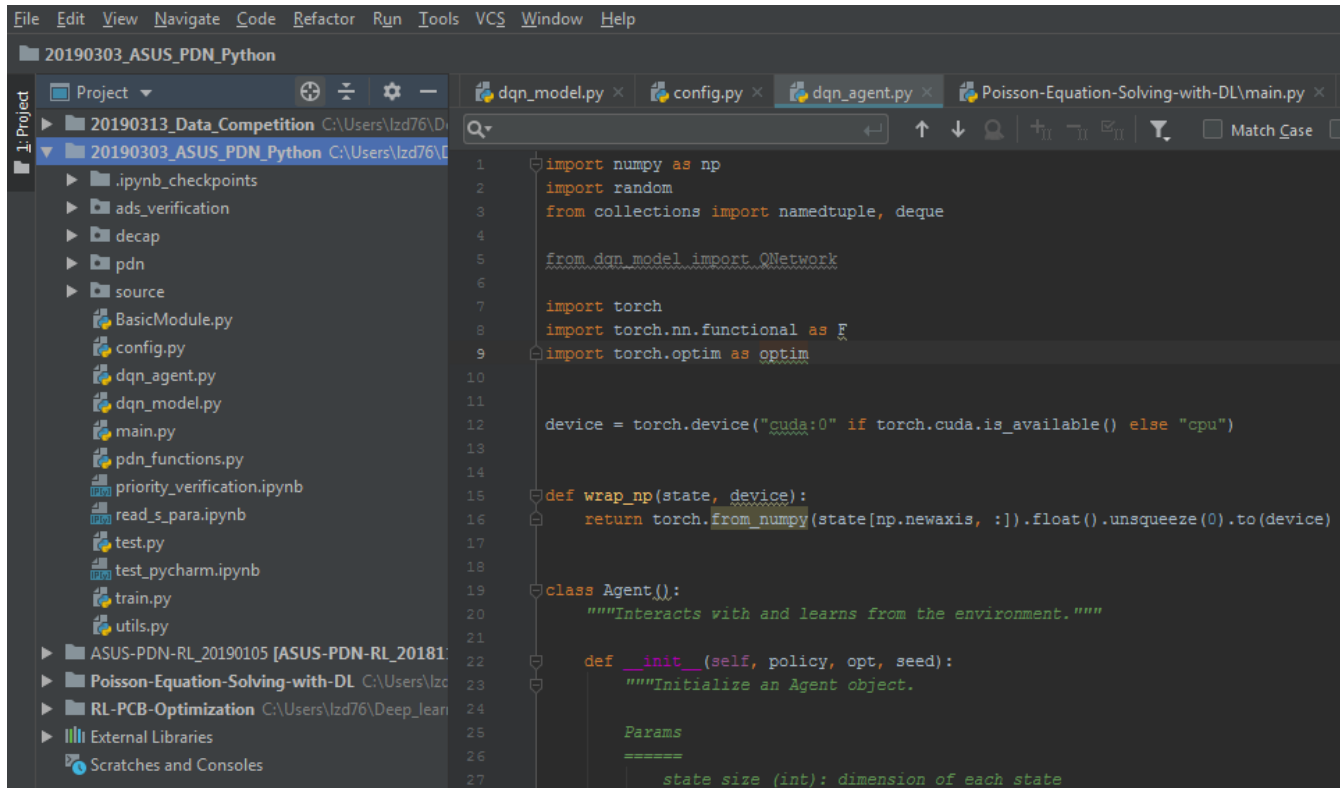
```
In [1]: a = 12
In [2]: a
Out[2]: 12
In [3]: b = a*2
In [4]: c = "Hello"
In [5]: c*2
Out[5]: 'HelloHello'
In [6]: %whos
```

Variable	Type	Data/Info
a	int	12
b	int	24
c	str	Hello

The final cell is an empty input prompt: `In []:`

PyCharm

<https://www.jetbrains.com/pycharm/>



The screenshot displays the PyCharm IDE interface. On the left, the 'Project' view shows a directory structure for '20190303_ASUS_PDN_Python'. The main editor window shows the code for 'dqn_agent.py'. The code includes imports for numpy, random, collections, torch, and torch.nn.functional. It defines a 'wrap_np' function and a 'class Agent()' with an 'init' method. The code is as follows:

```
1 import numpy as np
2 import random
3 from collections import namedtuple, deque
4
5 from dqn_model import QNetwork
6
7 import torch
8 import torch.nn.functional as F
9 import torch.optim as optim
10
11
12 device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
13
14
15 def wrap_np(state, device):
16     return torch.from_numpy(state[np.newaxis, :]).float().unsqueeze(0).to(device)
17
18
19 class Agent():
20     """Interacts with and learns from the environment."""
21
22     def __init__(self, policy, opt, seed):
23         """Initialize an Agent object.
24
25         Params
26         =====
27         state_size (int): dimension of each state
```

- PyCharm: easy to work with when working on a large project containing different Python files
- Can see intermediate results in debugging mode

Part I – Introduction to Python

- What is Python
- Commonly-used Python editors
- **Data Structures in Python**
- Control Structures (Selection and Loop)
- Python Functions
- Python Classes
- An Python package example: skrf

Overview of Data Structures in Python

▶ Basic Data Types in Python

- Numeric type
 - ▶ Integer
 - ▶ Float
 - ▶ Complex
 - ▶ Boolean
- Container
 - ▶ List (costless insertion and append)
 - ▶ String
 - ▶ Dictionary (key-value pairs for fast lookup)
 - ▶ Set
 - ▶ Tuple

▶ In NumPy package

- Multi-dimensional arrays (array oriented computing)

▶ In Pandas package (discuss later)

- Series
- DataFrame
- Panel

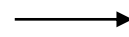
A container is simply an object that holds a collection of other objects.

Lists, Tuple, Set, Dictionary

- **List** is a collection which is ordered and changeable. Allows duplicate members.
- **Tuple** is a collection which is ordered and unchangeable. Allows duplicate members.
- **Set** is a collection which is unordered and unindexed. No duplicate members.
- **Dictionary** is a collection which is unordered, changeable and indexed. No duplicate members.

List:

```
>>> classmates = ['Michael', 'Bob', 'Tracy']
>>> classmates
['Michael', 'Bob', 'Tracy']
```



Usually used in 'for' loop

```
names = ['Michael', 'Bob', 'Tracy']
for name in names:
    print(name)
```

Tuple:

```
>>> classmates = ('Michael', 'Bob', 'Tracy')
```

Set:

```
>>> s = set([1, 2, 3])
>>> s
{1, 2, 3}
```

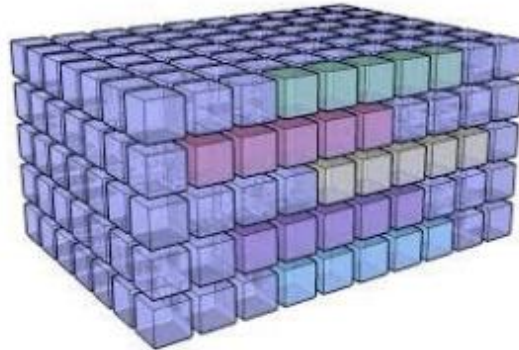
Dictionary:

```
>>> d = {'Michael': 95, 'Bob': 75, 'Tracy': 85}
>>> d['Michael']
95
```

Attention: Python index begins with 0 !!!

Use NumPy for Array/Matrix Oriented Computing

- ▶ NumPy is a fundamental package for scientific computing. It's an open source alternative to Matlab.
- ▶ It provides support for **homogeneous** multi-dimensional arrays and matrices, along with efficient mathematical functions for operating on these arrays.
 - The performance of NumPy is closer to hardware (efficiency).
 - NumPy is very convenient to use when we want to do math with a large set of numbers.



NumPy Overview

▶ Create Arrays

- Array
- Linspace
- Arange
- Other built-in functions

▶ Modify Arrays

- Append
- Reshape
- Transpose
- Sort/ArgSort
- Round

▶ Array Operations

- Max
- Min
- Dot
- Sum

▶ Indexing and Slicing

For a comprehensive reference of NumPy methods, refer to:

<https://docs.scipy.org/doc/numpy/reference/routines.html>

Part I – Introduction to Python

- What is Python
- Commonly-used Python editors
- Data Structures in Python
- **Control Structures (Selection and Loop)**
- Python Functions
- Python Classes
- An Python package example: skrf

If-elif-else Selection Structure in Python

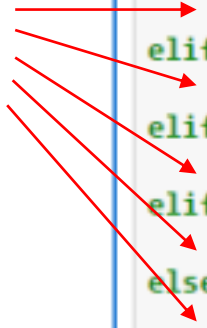
```
# Ask user to input score (a float number between 0 and 100)
score = float(input("Please input score (0-100):"))

Please input score (0-100):59.8

## Use if-elif-else statement to implement the multi-way selection structure.
if score >= 90:
    print("Grade = A")
elif score >= 80:
    print("Grade = B")
elif score >= 70:
    print("Grade = C")
elif score >= 60:
    print("Grade = D")
else:
    print("Grade = F")

Grade = F
```

Indentation
is necessary
in Python!



The Nearest Rule (if else ambiguity)

The else/elif clause matches the nearest preceding if/elif clause in the same block.

Iteration/Loop Structure

- ▶ Python provides three statements to support looping
 - `while` statement
 - `for` statement
- ▶ Two statements used to explicitly control looping
 - `break` statement
 - `continue` statement

While loop

```
# Ask user to input score (a float number between 0 and 100)
score = -1 # Initialize the score as -1

while score < 0 or score > 100:
    score = float(input("Please input score (0-100):"))

Please input score (0-100):-90
Please input score (0-100):-1
Please input score (0-100):111
Please input score (0-100):345
Please input score (0-100):-89
Please input score (0-100):101
Please input score (0-100):99
```

```
# Use a while loop to sum numbers from 1 to n till the sum >= 100.

sum = 0 # Initialize the sum
i = 1 # Initialize the number to sum

while i <= 100:
    sum = sum + i
    if sum >= 100:
        break
    i = i + 1

print("The sum is:", sum)
print("i =", i)
```

break

```
# Use a while loop to sum numbers from 1 to 20, bypassing 10 and 11.

sum = 0 # Initialize the sum
i = 0 # Initialize the number to sum

while i < 20:
    i = i + 1
    if i == 10 or i == 11:
        continue
    sum = sum + i

print("The sum is:", sum)
print("i =", i)
```

continue

Part I – Introduction to Python

- What is Python
- Commonly-used Python editors
- Data Structures in Python
- Control Structures (Selection and Loop)
- **Python Functions**
- Python Classes
- An Python package example: skrf

Function Definition in Python

- General syntax:

```
def function_name(parameters):  
    block of statements
```

```
def my_abs(x):  
    if x >= 0:  
        return x  
    else:  
        return -x
```

- Lambda expression:

```
lambda input_parameters: returned_expression
```

- Lambda functions are usually used temporally without a formal name.

```
x = lambda a : a + 10  
print(x(5))
```

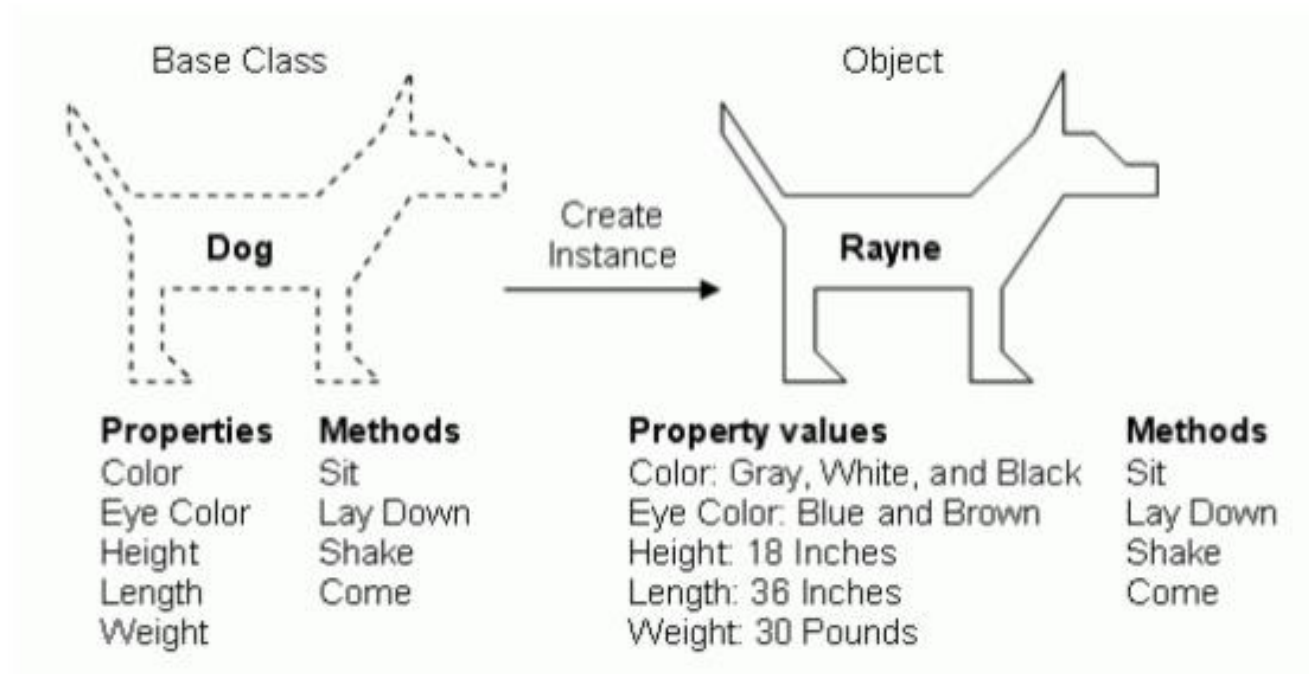
```
x = lambda a, b : a * b  
print(x(5, 6))
```

Part I – Introduction to Python

- What is Python
- Commonly-used Python editors
- Data Structures in Python
- Control Structures (Selection and Loop)
- Python Functions
- **Python Classes**
- An Python package example: skrf

Python: Object-based Language

Defining a Class



- Matlab is a function-based programming language.
- Differently, Python is a object-based language. We can define objects under classes. Then and the properties and methods (functions) can be easily called.
- Also, for Python, many packages online well-written by others are free to download and be installed.

Python Class Definition

Example

Create a class named MyClass, with a property named x:

```
class MyClass:  
    x = 5
```

Create an object named p1, and print the value of x:

```
p1 = MyClass()  
print(p1.x)
```

```
class Person:  
    def __init__(self, name, age):  
        self.name = name  
        self.age = age  
  
    def myfunc(self):  
        print("Hello my name is " + self.name)  
  
p1 = Person("John", 36)  
p1.myfunc()
```

properties

methods

- The **__init__()** function is called automatically every time the class is being used to create a new object.
- The **self** parameter is a reference to the current instance of the class, and is used to access variables that belongs to the class.
- It does not have to be named self , you can call it whatever you like, but it has to be the first parameter of any function in the class.
- There are other properties of Python class such as **inheritance**, which will not be covered in detail here.

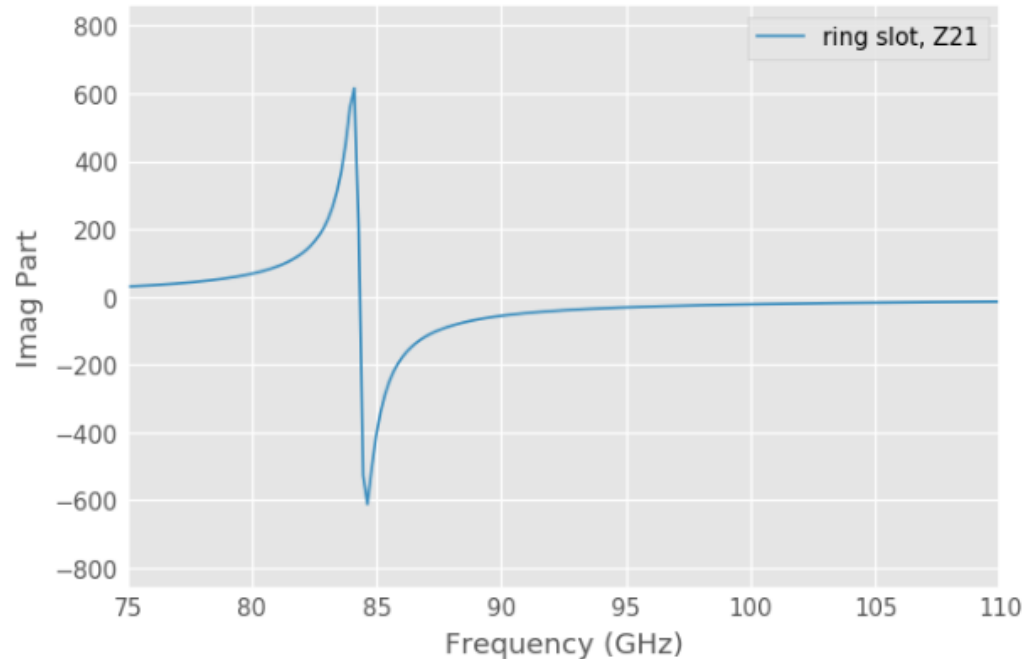
Part I – Introduction to Python

- What is Python
- Commonly-used Python editors
- Data Structures in Python
- Control Structures (Selection and Loop)
- Python Functions
- Python Classes
- An Python package example: skrf

RF Library in Python

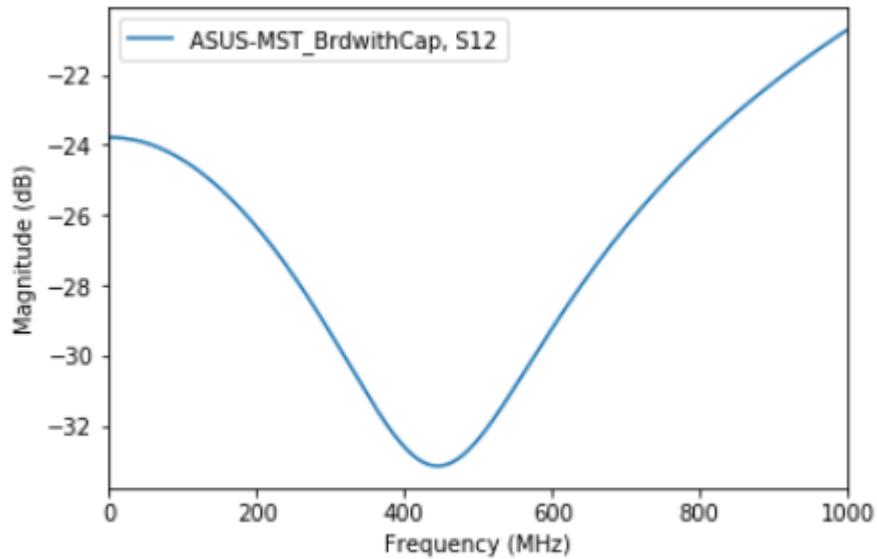
- Found a library in Python called **skrf**, which is free to install and use. (<https://scikit-rf.readthedocs.io/en/latest/tutorials/Networks.html#Introduction>)
- This library can handle S-parameters and Z-parameters.

- ▢ Networks
 - Introduction
 - Creating Networks
 - Basic Properties
 - Slicing
 - Plotting
- ▢ Operators
 - Connecting Multi-ports
 - Interpolation and Concatenation
 - Reading and Writing
 - Other Parameters
 - Conclusion
 - References



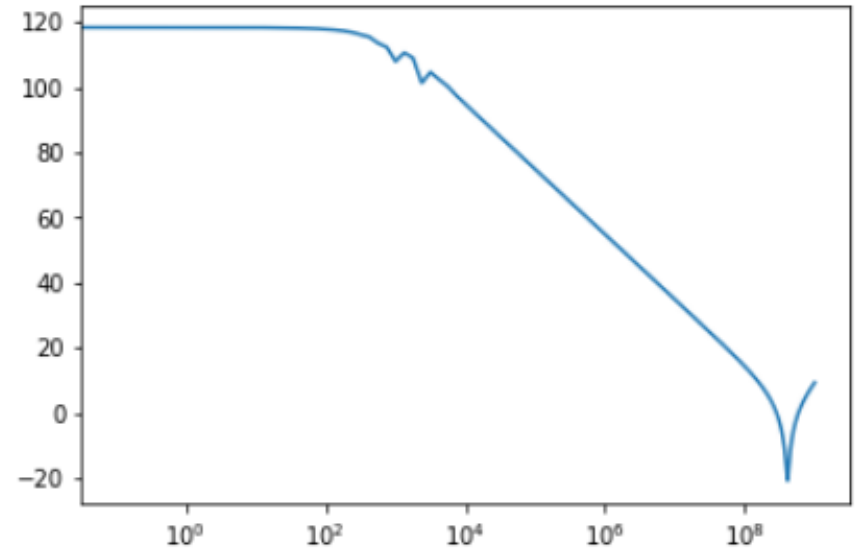
Plotting

```
# plot S-parameter  
asus_brd.plot_s_db(m=0,n=1)
```



Obtain frequency Obtain Z-para

```
# plot Z11 in log scale  
plt.semilogx(asus_brd.frequency.f, asus_brd.z_db[:,0,0])  
plt.show()
```



Connecting Multi-ports

```
merged_network
```

```
16-Port Network: 'ASUS-MST_BrdwithCap', 1.0-1000.0 MHz, 201 pts,  
0.+0.j 50.+0.j  
50.+0.j 50.+0.j 50.+0.j 50.+0.j 50.+0.j 50.+0.j 50.+0.j]
```

```
decap1_short
```

```
1-Port Network: 'GRM153R60G105ME95', 1.0-1000.0 MHz, 201 pts, z0:
```

```
new_net = rf.network.connect(merged_network,1,decap1_short,0)  
new_net
```

Connect two ports of two network

```
15-Port Network: 'ASUS-MST_BrdwithCap', 1.0-1000.0 MHz, 201 pts,  
0.+0.j 50.+0.j  
50.+0.j 50.+0.j 50.+0.j 50.+0.j 50.+0.j 50.+0.j]
```

Inner connect two ports of one network

```
asus_brd_mergeIC = rf.network.innerconnect(asus_brd,0,1)
```

```
asus_brd_mergeIC 31-port => 29-port
```

```
29-Port Network: 'ASUS-MST_BrdwithCap', 0.0-1000.0 MHz, 411 pts, z0=[50  
0.+0.j 50.+0.j  
50.+0.j 50.+0.j 50.+0.j 50.+0.j 50.+0.j 50.+0.j 50.+0.j 50.+0.j 50.+0.j  
50.+0.j 50.+0.j 50.+0.j 50.+0.j 50.+0.j 50.+0.j 50.+0.j 50.+0.j 50.+0.j  
50.+0.j 50.+0.j]
```

Summary of Part I

- Python is a high-level programming language which is free and open-source, and convenient to use.
- When you want some functions, try to search and download existing packages online first.

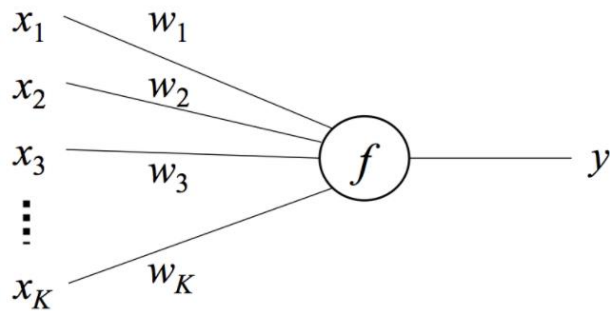
Part II – Introduction to Machine Learning

- Brief introduction to neural network and deep learning
- How neural network is trained – gradient descent and backpropagation
- A simple tutorial of using Pytorch for machine learning programming in Python

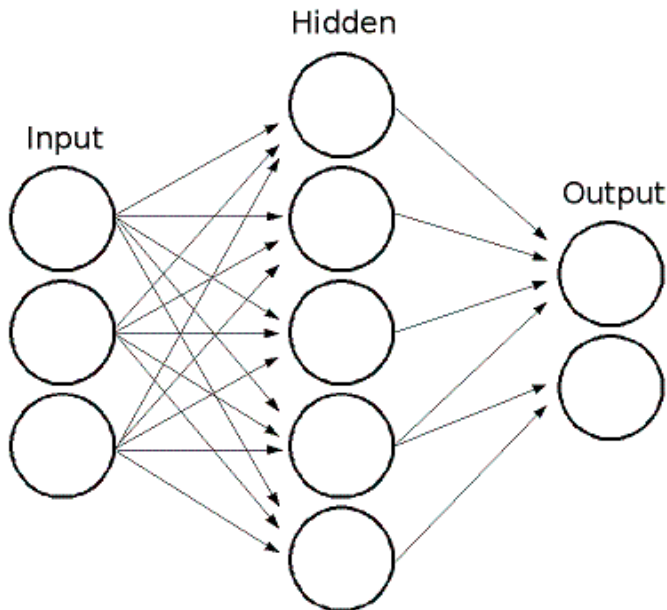
Part II – Introduction to Machine Learning

- Brief introduction to neural network and deep learning
- How neural network is trained – gradient descent and backpropagation
- A simple tutorial of using Pytorch for machine learning programming in Python

Neural Network (NN)



$$y = f\left(\sum_{i=1}^K w_i x_i\right) = f(\mathbf{w}^T \mathbf{x})$$



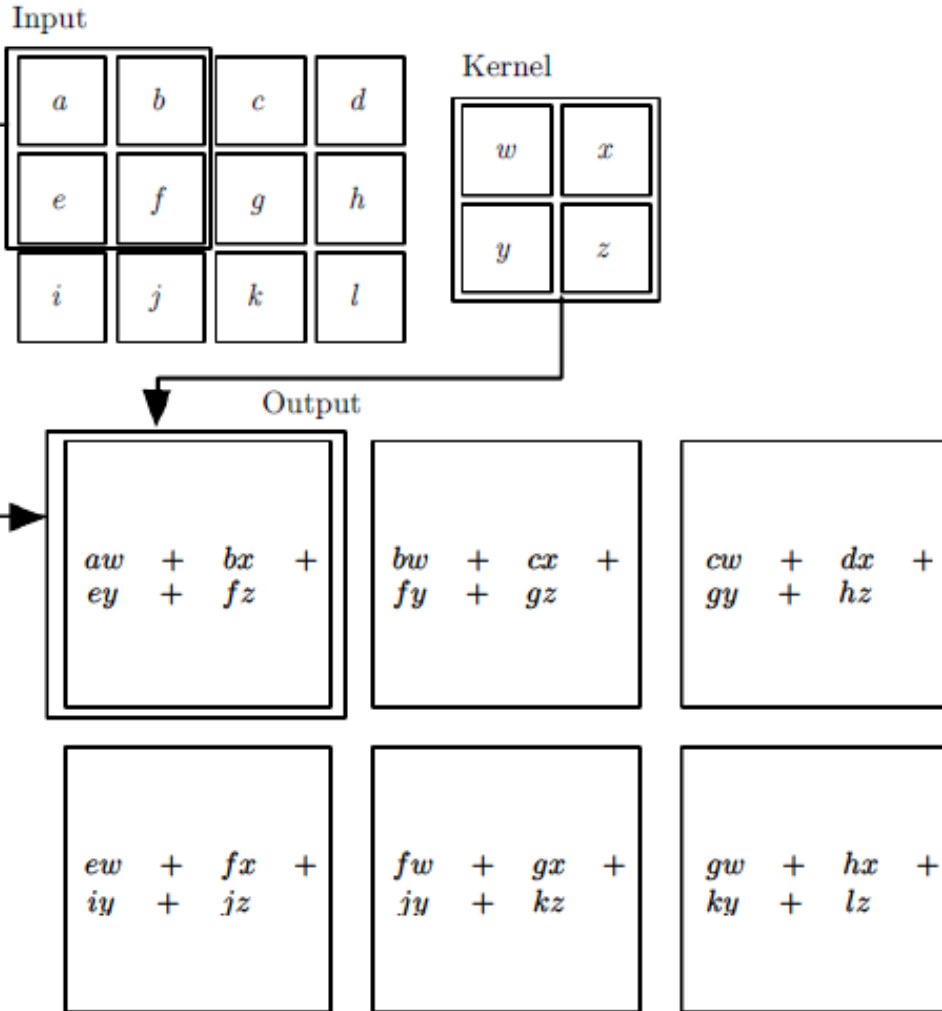
- The structure of neural network is similar to neuron structure in human brain.
- The model has an input layer, an output layer and an arbitrary number of hidden layers.
- Each neuron can be regarded as a nonlinear function (activation function) of the weighted sum of its inputs.
- Pros:
 - High accuracy, capacity and robustness.
 - Can achieve complex non-linearity.
- Cons:
 - Need large data size
 - Need high computational ability.
 - Black-box model and hard to understand.
- Applications: regression and classification.
- How can neural network be trained?

Image Recognition



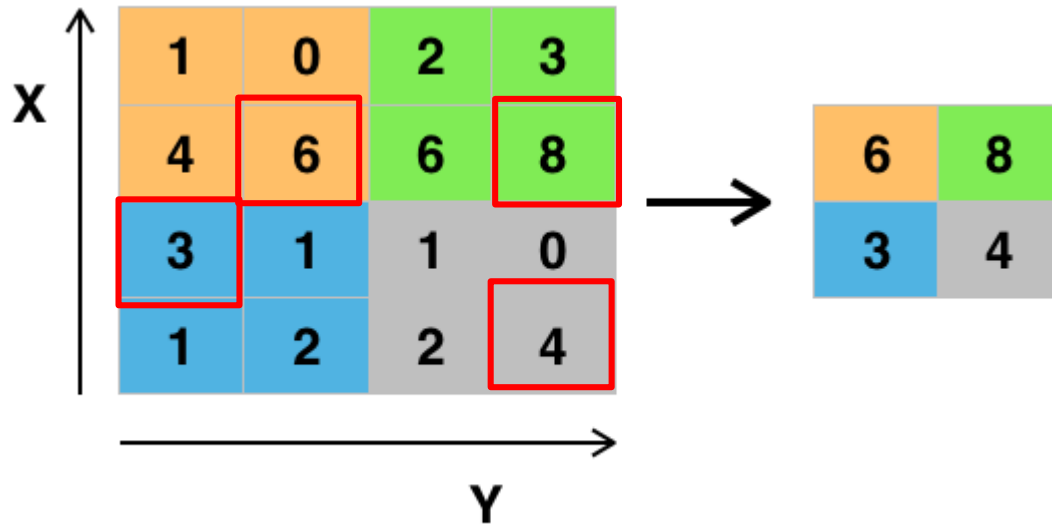
- For human beings, it is intuitive and easy to recognize the kid in the picture. Even if the position and environment changes, we don't need to learn the concept again.
- But for computers, it is hard to recognize it if the picture changes for a little bit. And it is hard to build a formal rule about how a kid looks like.
- Intuitively, we can feel there exists some hierarchy or conceptual structure in the picture, like from pixels to edges, and ears, mouth, nose, head and etc. Human beings recognize pictures mainly by high-level features, not by individual pixels.

2D Convolution



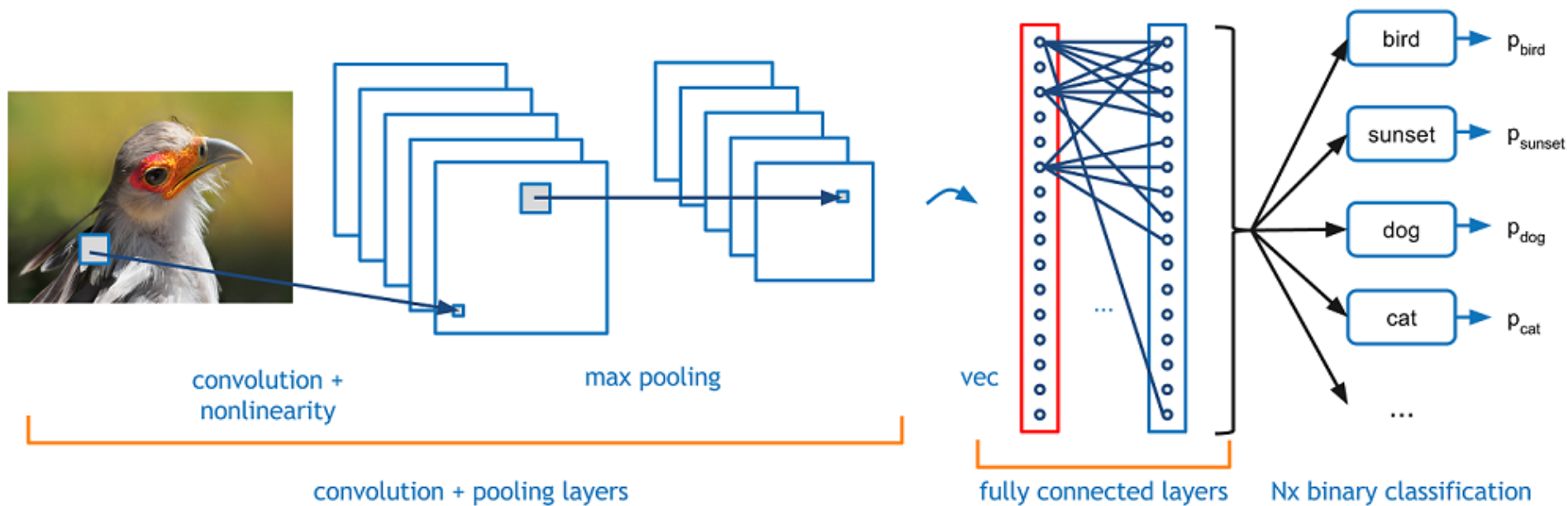
- A window smaller than image size slides through the image and multiply with each small area it goes through.
- We can detect **small and meaningful features** such as edges with kernels that occupy only tens or hundreds of pixels.
- Using a couple of convolutional layers in series helps extract more complex features.
- This also reduces the memory requirements of the model and improve its statistical efficiency.

Max Pooling



- Max pooling partitions the input image into a set of non-overlapping rectangles and, for each such sub-region, outputs the maximum.
- The pooling layer serves to progressively reduce the spatial size of the representation, to reduce the number of parameters and amount of computation in the network, and hence to also control overfitting.
- It is common to periodically insert a pooling layer between successive convolutional layers in a CNN architecture.

Convolutional Neural Network (CNN)



- A CNN consists of an input and an output layer, as well as multiple hidden layers. The hidden layers of a CNN typically consist of convolutional layers, pooling layers, fully connected layers and normalization layers.
- A CNN contains so many parameters that it needs large amounts of data to be trained well. Usually at least millions of images are required to train a good image classifier like Google. In deep learning, having more data is more significant than having a better model!

Part II – Introduction to Machine Learning

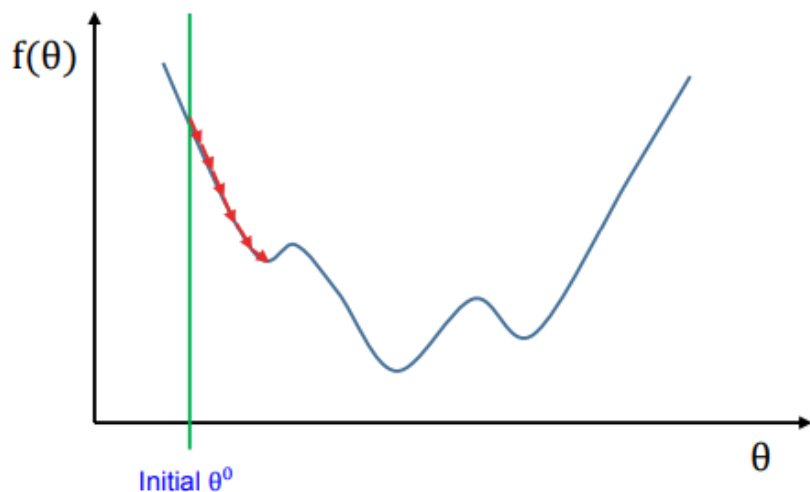
- Brief introduction to neural network and deep learning
- How neural network is trained – gradient descent and backpropagation
- A simple tutorial of using Pytorch for machine learning programming in Python

Optimization Method – Gradient Descent

- When there is no closed form, we have to use computational approach such as gradient descent.

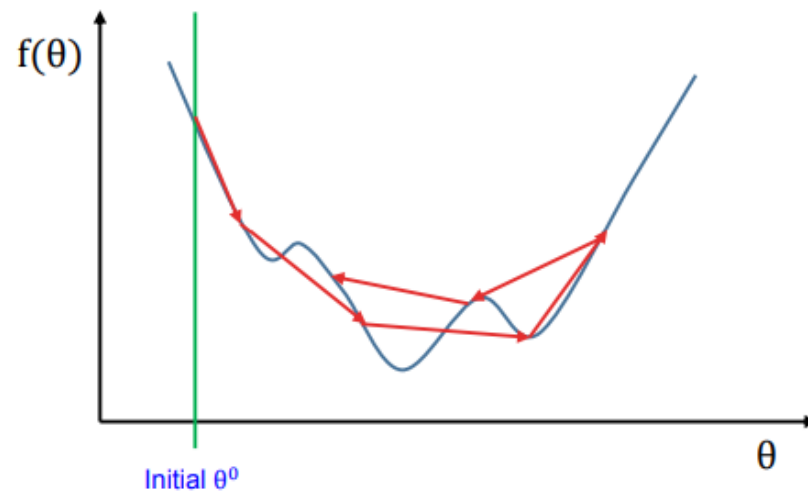
$$\theta^1 = \theta^0 - \alpha \nabla f(\theta^0)$$

The cost function must be differentiable;
 θ is updated in the *opposite* direction of the gradient.



Learning rate is too small:

- Convergence is very slow;
- May converge at a local minimum.



Learning rate is too large:

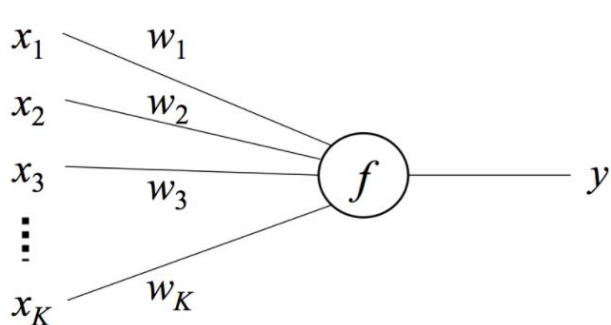
- May fluctuate around the minimum.

Training Method: Backpropagation (1 / 2)

Review:

- **Chain rule of calculus:** $\frac{dz}{dx} = \frac{dz}{dy} \frac{dy}{dx}$
- **Gradient descent** method to update weights: $\theta_j := \theta_j - \eta \cdot \frac{\partial}{\partial \theta_j} E(\theta)$

where η is learning rate, and $E(\theta)$ is loss function that needs to be minimized.



$$y = f\left(\sum_{i=0}^K w_i x_i\right) = f(\mathbf{w}^T \mathbf{x})$$

$$f(u) = \frac{1}{1 + e^{-u}} \quad (\text{logistic function})$$

Derivative of $f(u)$: $\frac{df(u)}{du} = f(u)(1 - f(u)) = f(u)f(-u)$

Loss function: $E = \frac{1}{2}(t - y)^2 = \frac{1}{2}(t - f(\mathbf{w}^T \mathbf{x}))^2$

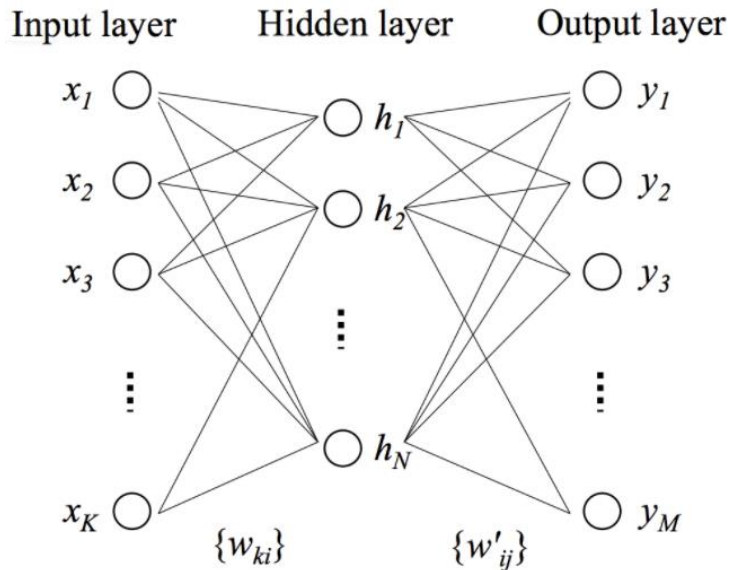
$$\begin{aligned} \frac{\partial E}{\partial w_i} &= \frac{\partial E}{\partial y} \cdot \frac{\partial y}{\partial u} \cdot \frac{\partial u}{\partial w_i} \\ &= (y - t) \cdot y(1 - y) \cdot x_i \end{aligned}$$

t : actual output



$$\mathbf{w}^{new} = \mathbf{w}^{old} - \eta \cdot (y - t) \cdot y(1 - y) \cdot \mathbf{x}$$

Training Method: Backpropagation (2 / 2)



$$h_i = f(u_i) = f\left(\sum_{k=1}^K w_{ki} x_k\right)$$

$$y_j = f(u'_j) = f\left(\sum_{i=1}^N w'_{ij} h_i\right)$$

$$E = \frac{1}{2} \sum_{j=1}^M (y_j - t_j)^2$$

$$\frac{\partial E}{\partial w'_{ij}} = \frac{\partial E}{\partial y_j} \cdot \frac{\partial y_j}{\partial u'_j} \cdot \frac{\partial u'_j}{\partial w'_{ij}} = (y_j - t_j) \cdot y_j(1 - y_j) \cdot h_i$$

$$w'_{ij}{}^{new} = w'_{ij}{}^{old} - \eta \cdot (y_j - t_j) \cdot y_j(1 - y_j) \cdot h_i$$

$$\frac{\partial E}{\partial w_{ki}} = \sum_{j=1}^M \left(\frac{\partial E}{\partial y_j} \cdot \frac{\partial y_j}{\partial u'_j} \cdot \frac{\partial u'_j}{\partial h_i} \right) \cdot \frac{\partial h_i}{\partial u_i} \cdot \frac{\partial u_i}{\partial w_{ki}} = \sum_{j=1}^M [(y_j - t_j) \cdot y_j(1 - y_j) \cdot w'_{ij}] \cdot h_i(1 - h_i) \cdot x_k$$

$$w_{ki}{}^{new} = w_{ki}{}^{old} - \eta \cdot \sum_{j=1}^M [(y_j - t_j) \cdot y_j(1 - y_j) \cdot w'_{ij}] \cdot h_i(1 - h_i) \cdot x_k$$

Part II – Introduction to Machine Learning

- Brief introduction to neural network and deep learning
- How neural network is trained – gradient descent and backpropagation
- A simple tutorial of using Pytorch for machine learning programming in Python

Example of Pytorch – Linear Regression (1/3)

https://github.com/yunjey/pytorch-tutorial/blob/master/tutorials/01-basics/linear_regression/main.py

```
1 import torch
2 import torch.nn as nn
3 import numpy as np
4 import matplotlib.pyplot as plt
```

Import modules

```
5
6
7 # Hyper-parameters
8 input_size = 1
9 output_size = 1
10 num_epochs = 60
11 learning_rate = 0.001
```

Define hyper-parameters

```
12
13 # Toy dataset
14 x_train = np.array([[3.3], [4.4], [5.5], [6.71], [6.93], [4.168],
15                    [9.779], [6.182], [7.59], [2.167], [7.042],
16                    [10.791], [5.313], [7.997], [3.1]], dtype=np.float32)
17
18 y_train = np.array([[1.7], [2.76], [2.09], [3.19], [1.694], [1.573],
19                    [3.366], [2.596], [2.53], [1.221], [2.827],
20                    [3.465], [1.65], [2.904], [1.3]], dtype=np.float32)
```

Training dataset

Example of Pytorch – Linear Regression (2/3)

```
# Linear regression model
model = nn.Linear(input_size, output_size)

# Loss and optimizer
criterion = nn.MSELoss()
optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate)

# Train the model
for epoch in range(num_epochs):
    # Convert numpy arrays to torch tensors
    inputs = torch.from_numpy(x_train)
    targets = torch.from_numpy(y_train)

    # Forward pass
    outputs = model(inputs)
    loss = criterion(outputs, targets)

    # Backward and optimize
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

    if (epoch+1) % 5 == 0:
        print ('Epoch [{} / {}], Loss: {:.4f}'.format(epoch+1, num_epochs, loss.item()))
```

Directly use `nn.linear` module

Use `nn.MSELoss()` function as objective function

Use `SGD` optimizer

Create a `Tensor` from `numpy` array

Calculate output

Calculate loss

Calculate gradient and update parameters

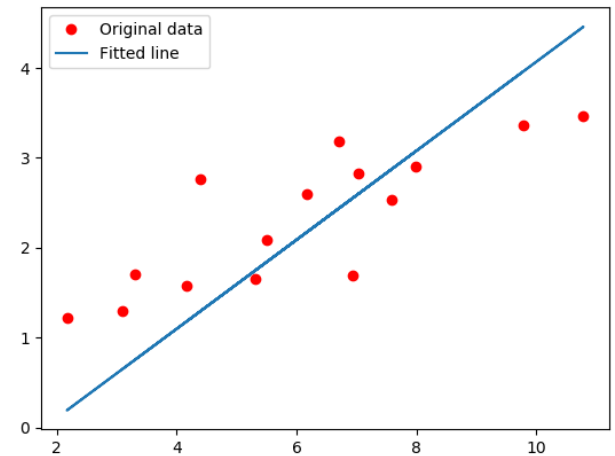
Example of Pytorch – Linear Regression (3/3)

model has been trained

```
# Plot the graph
predicted = model(torch.from_numpy(x_train)).detach().numpy()
plt.plot(x_train, y_train, 'ro', label='Original data')
plt.plot(x_train, predicted, label='Fitted line')
plt.legend()
plt.show()
```

```
# Save the model checkpoint
torch.save(model.state_dict(), 'model.ckpt')
```

Save the trained model



- When using Python for machine learning applications, we don't need to rewrite the gradient descent and backpropagation algorithm by ourselves.
- We just need to use the existing modules in the packages such as PyTorch, Tensorflow, which makes it convenient.

Example of Pytorch – CNN

- Complete code can be found at: https://github.com/yunjey/pytorch-tutorial/blob/master/tutorials/02-intermediate/convolutional_neural_network/main.py

```
# Convolutional neural network (two convolutional layers)
class ConvNet(nn.Module):
    def __init__(self, num_classes=10):
        super(ConvNet, self).__init__()
        self.layer1 = nn.Sequential(
            nn.Conv2d(1, 16, kernel_size=5, stride=1, padding=2),
            nn.BatchNorm2d(16),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2))
        self.layer2 = nn.Sequential(
            nn.Conv2d(16, 32, kernel_size=5, stride=1, padding=2),
            nn.BatchNorm2d(32),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2))
        self.fc = nn.Linear(7*7*32, num_classes)

    def forward(self, x):
        out = self.layer1(x)
        out = self.layer2(out)
        out = out.reshape(out.size(0), -1)
        out = self.fc(out)
        return out
```

```
# Loss and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)

# Train the model
total_step = len(train_loader)
for epoch in range(num_epochs):
    for i, (images, labels) in enumerate(train_loader):
        images = images.to(device)
        labels = labels.to(device)

        # Forward pass
        outputs = model(images)
        loss = criterion(outputs, labels)

        # Backward and optimize
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

    if (i+1) % 100 == 0:
        print ('Epoch [{} / {}], Step [{} / {}], Loss: {:.4f}'
              .format(epoch+1, num_epochs, i+1, total_step, loss.item()))
```

- Class inheritance
- Self-defined neural network

Example of Tensorflow

- Can refer to this tutorial on GitHub
 - <https://github.com/aymericdamien/TensorFlow-Examples>

Summary of Part II

- There are many available code example or template on [GitHub](#), which can be downloaded freely.
- You just need to slightly modify the code to fit your desired applications.



Questions?