TensorFlow

Deep Learning Framework

Quang-Vinh Dinh
Ph.D. in Computer Science

Outline

- Introduction to Tensorflow
- Introduction to Tensforflow.Keras
- Model Construction
- Model Training and Inference
- Applying Softmax for Image Data

Introduction

Developer(s)	Google Brain Team[1]
Initial release	November 9, 2015; 6 years ago
Stable release	2.6.1[2] (1 November 2021; 30 days ago) / May 14, 2021; 6 months ago
Repository	github.com/tensorflow/tensorflow
Written in	Python, C++, CUDA
Platform	Linux, macOS, Windows, Android, JavaScript[3]
Type	Machine learning library
License	Apache License 2.0
Website	www.tensorflow.org

Installation

pip install tensorflow

Package Declaration

import tensorflow as tf

Example 1

```
1  import tensorflow as tf
2
3  print("TensorFlow version: ", tf.__version__)
4  print("Keras version: ", tf.keras.__version__)
```

```
TensorFlow version:  2.7.0
Keras version:  2.7.0
```

Example 2

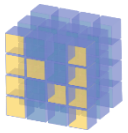
```
1  import tensorflow as tf
2
3  if tf.test.is_gpu_available():
4      print('Running on GPU')
5  else:
6      print('Running on CPU')
```

```
Running on CPU
```

Introduction

❖ Tensor

- ❖ ~ ndarray in Numpy
- ❖ Run on both CPU and GPU
- ❖ All tensors are immutable
- ❖ Multi-dimensional arrays with a uniform type




```python
import tensorflow as tf
import numpy as np

# create a ndarray
an_array = np.array([1,3,5,7,11])
print(type(an_array), an_array)

# convert from ndarray to tensor
a_tensor = tf.convert_to_tensor(an_array)
print(a_tensor)

# convert from tensor to ndarray
array_2 = a_tensor.numpy()
print(type(array_2), array_2)
```

```
<class 'numpy.ndarray'> [ 1  3  5  7 11]
tf.Tensor([ 1  3  5  7 11], shape=(5,), dtype=int32)
<class 'numpy.ndarray'> [ 1  3  5  7 11]
```

Important attributes

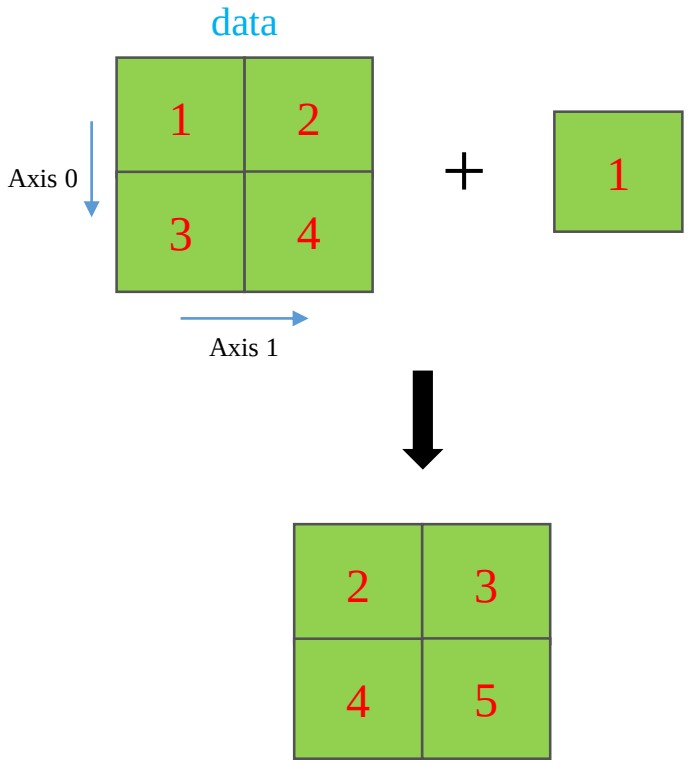
```python
import tensorflow as tf
import numpy as np

# create a 2x3 aray
an_array = np.array([[1,2],
                     [3,4],
                     [5,6]])

# convert to tensor
a_tensor = tf.convert_to_tensor(an_array)

# no. of elements on each axis
print(a_tensor.shape)
# data type
print(a_tensor.dtype)

# no. of axises
print(a_tensor.ndim)
```

```
(3, 2)
<dtype: 'int32'>
2
```

Introduction

❖ Tensor

❖Broadcasting



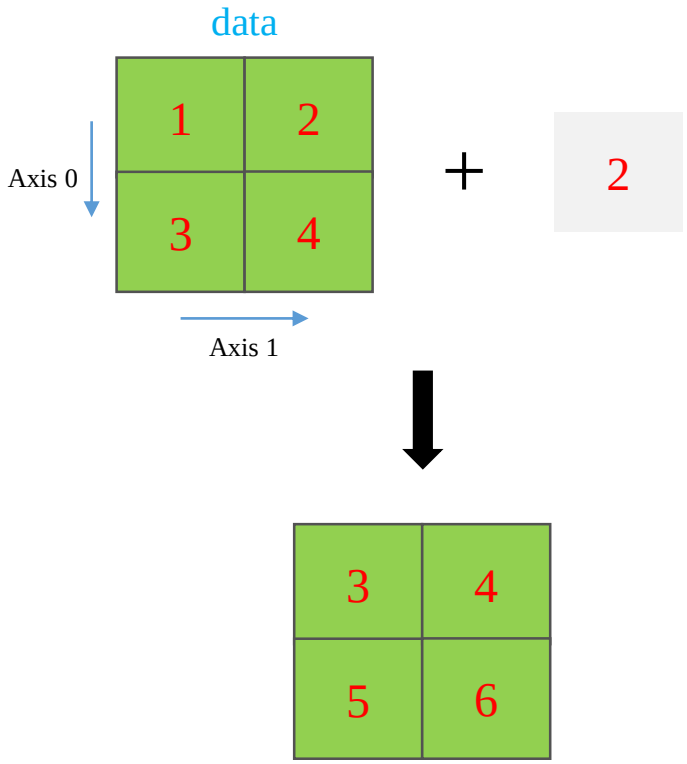

```python
import tensorflow as tf
import numpy as np

# create two tensors
tensor_1 = tf.convert_to_tensor([[1,2],
                                 [3,4]])
tensor_2 = tf.convert_to_tensor([1])

# add two tensors
tensor_3 = tensor_1 + tensor_2

print('tensor_1: \n', tensor_1)
print('tensor_2: \n', tensor_2)
print('tensor_3: \n', tensor_3)
```

```
tensor_1:
 tf.Tensor(
[[1 2]
 [3 4]], shape=(2, 2), dtype=int32)
tensor_2:
 tf.Tensor([1], shape=(1,), dtype=int32)
tensor_3:
 tf.Tensor(
[[2 3]
 [4 5]], shape=(2, 2), dtype=int32)
```

Introduction

❖ Tensor

❖Broadcasting



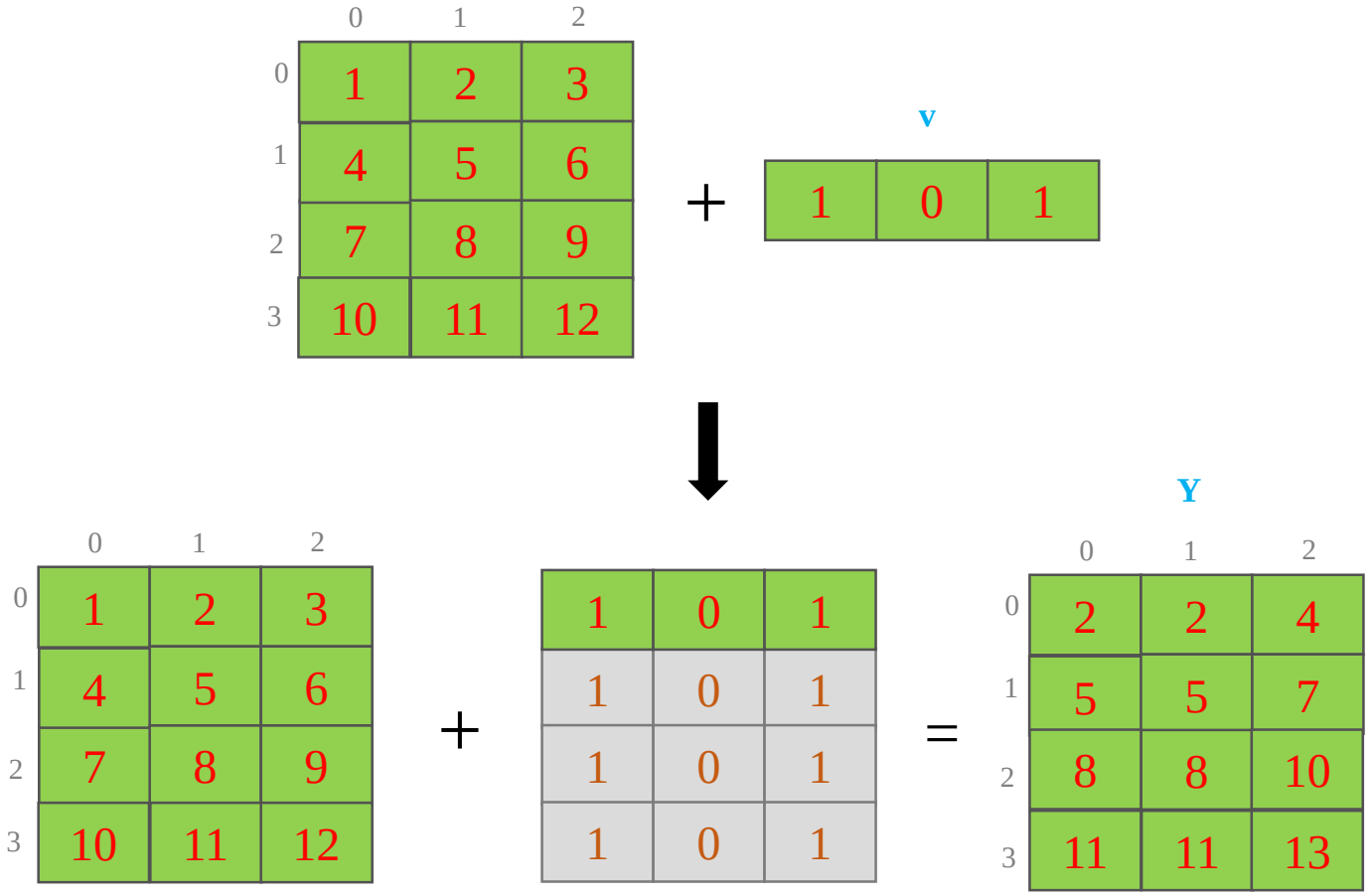

```python
import tensorflow as tf
import numpy as np

# create 2x2 tensor
tensor_1 = tf.convert_to_tensor([[1,2],
                                 [3,4]])

# add a number to a tensor
tensor_2 = tensor_1 + 2

print('tensor_1: \n', tensor_1)
print('tensor_2: \n', tensor_2)
```

```
tensor_1:
 tf.Tensor(
[[1 2]
 [3 4]], shape=(2, 2), dtype=int32)
tensor_2:
 tf.Tensor(
[[3 4]
 [5 6]], shape=(2, 2), dtype=int32)
```

Introduction

❖ Tensor

❖Broadcasting




```
1  import tensorflow as tf
2  import numpy as np
3
4  # create 4x3 tensor
5  np_data = np.array(range(1, 13)).reshape(4, 3)
6  tensor_1 = tf.convert_to_tensor(np_data)
7
8  # create the second tensor
9  tensor_2 = tf.convert_to_tensor([1, 0, 1])
10
11 # add a number to a tensor
12 tensor_3 = tensor_1 + tensor_2
13
14 print('tensor_1: \n', tensor_1)
15 print('tensor_2: \n', tensor_2)
16 print('tensor_3: \n', tensor_3)
```

```
tensor_1:
 tf.Tensor(
[[ 1  2  3]
 [ 4  5  6]
 [ 7  8  9]
 [10 11 12]], shape=(4, 3), dtype=int32)
tensor_2:
 tf.Tensor([1 0 1], shape=(3,), dtype=int32)
tensor_3:
 tf.Tensor(
[[ 2  2  4]
 [ 5  5  7]
 [ 8  8 10]
 [11 11 13]], shape=(4, 3), dtype=int32)
```

Introduction

❖ Tensor

❖Important functions

Squared Difference

$$sd = (x - y)^2$$

x	$(x-y)^2$ =	sd
1		16
2		9
3		4
4		1

```python
import tensorflow as tf

# create a list
x = [1,2, 3, 4]
y = 5

# compute squared difference
sd = tf.math.squared_difference(x,y)
print(sd)
```

```
tf.Tensor([16  9  4  1], shape=(4,), dtype=int32)
```

```python
import tensorflow as tf

# create a tensor
x = tf.convert_to_tensor([1,2, 3, 4])
y = 5

# compute squared difference
sd = tf.math.squared_difference(x,y)
print(sd)
```

```
tf.Tensor([16  9  4  1], shape=(4,), dtype=int32)
```

```python
import tensorflow as tf
import numpy as np

# create an ndarray
x = np.array([1,2, 3, 4])
y = 5

# compute squared difference
sd = tf.math.squared_difference(x,y)
print(sd)
```

```
tf.Tensor([16  9  4  1], shape=(4,), dtype=int32)
```

Introduction

❖ Tensor

❖Important functions

random.normal()

```
1  import tensorflow as tf
2
3  rand = tf.random.normal(shape = (3,2), mean=0, stddev=1)
4  print(rand)
```

```
tf.Tensor(
[[ 0.47103247 -0.12765862]
 [-0.26556632 -0.05912822]
 [ 1.0851953  -0.55289406]], shape=(3, 2), dtype=float32)
```

random.uniform()

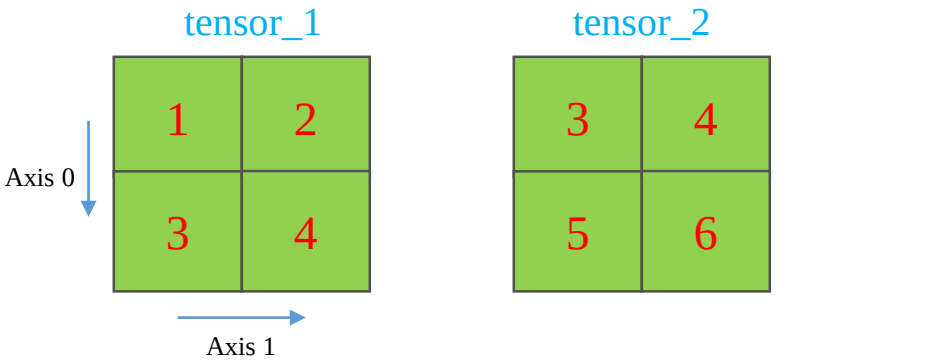
```
1  import tensorflow as tf
2
3  rand = tf.random.uniform(shape=(3,2), minval=0,
4                           maxval=9, dtype=tf.int32)
5  print(rand)
```

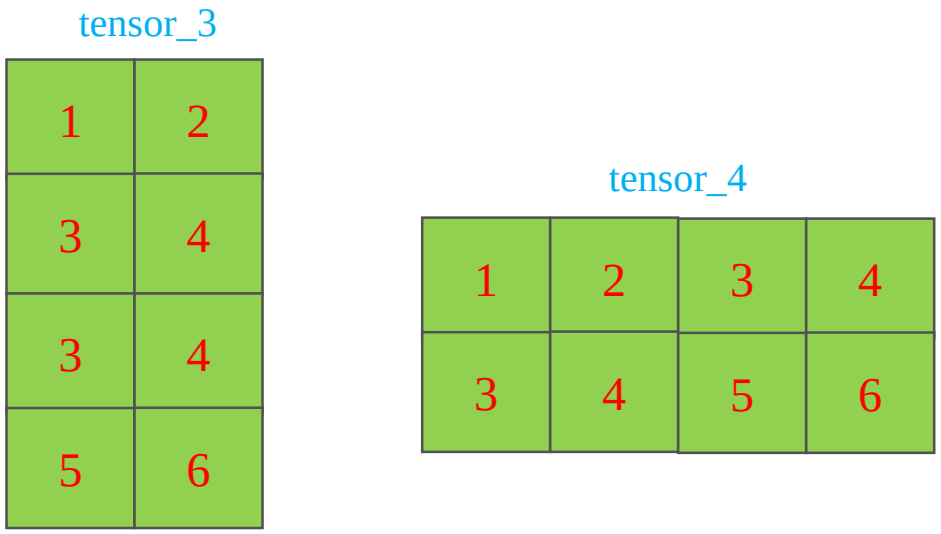
```
tf.Tensor(
[[4 0]
 [1 8]
 [0 2]], shape=(3, 2), dtype=int32)
```

Introduction

❖ Tensor

❖Important functions





concat()

```python
import tensorflow as tf

tensor_1 = tf.random.normal(shape=(2,2), mean=0, stddev=1)
tensor_2 = tf.random.normal(shape=(2,2), mean=0, stddev=1)

# concat two tensors along axis_0
tensor_3 = tf.concat([tensor_1, tensor_1], axis=0)

# concat two tensors along axis_1
tensor_4 = tf.concat([tensor_1, tensor_1], axis=1)

print(tensor_1.shape)
print(tensor_2.shape)
print(tensor_3.shape)
print(tensor_4.shape)
```

```
(2, 2)
(2, 2)
(4, 2)
(2, 4)
```

Introduction

❖ Tensor

❖ Important functions

1	2
7	9
8	6

.argmin(axis=0) = | 0 | 0 |

1	2
7	9
8	6

.argmin(axis=1) = | 0 | 0 | 1 |

argmin()

```python
import tensorflow as tf

# create a tensor
tensor = tf.random.uniform(shape=(3,6), minval=0,
                           maxval=20, dtype=tf.int32)

# find the index of the min value
min_position_1 = tf.argmin(tensor, axis=0)
min_position_2 = tf.argmin(tensor, axis=1)

print(tensor)
print('min_position_1: ', min_position_1)
print('min_position_2: ', min_position_2)
```

```
tf.Tensor(
[[ 5  0 13 11  4 10]
 [19  6  7  5 17  6]
 [18  9  9  2  1 11]], shape=(3, 6), dtype=int32)
min_position_1:  tf.Tensor([0 0 1 2 2 1], shape=(6,), dtype=int64)
min_position_2:  tf.Tensor([1 3 4], shape=(3,), dtype=int64)
```

Introduction

❖ Tensor

❖ Important functions

1	2
7	9
8	6

.argmax(axis=0) = | 2 | 1 |

1	2
7	9
8	6

.argmax(axis=1) = | 1 | 1 | 0 |

argmax()

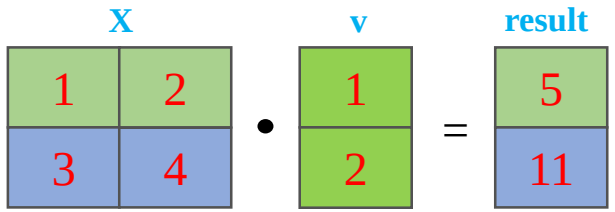
```
1  import tensorflow as tf
2
3  # create a tensor
4  tensor = tf.random.uniform(shape=(3,6), minval=0,
5                             maxval=20, dtype=tf.int32)
6
7  # find the index of the max value
8  max_position_1 = tf.argmax(tensor, axis=0)
9  max_position_2 = tf.argmax(tensor, axis=1)
10
11 print(tensor)
12 print('max_position_1: ', max_position_1)
13 print('max_position_2: ', max_position_2)
```

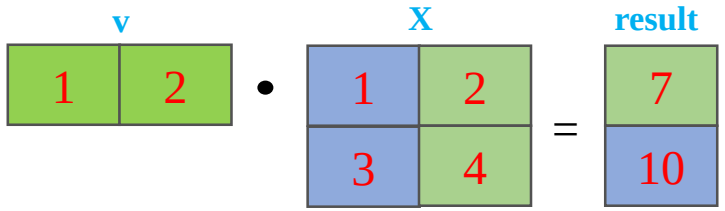
```
tf.Tensor(
[[ 8 10 13  1 12  1]
 [ 8 19  5  3 16 16]
 [ 2 16  7  1  4 17]], shape=(3, 6), dtype=int32)
max_position_1:  tf.Tensor([0 1 0 1 1 2], shape=(6,), dtype=int64)
max_position_2:  tf.Tensor([2 1 5], shape=(3,), dtype=int64)
```

Introduction

❖ Tensor

❖ Important functions





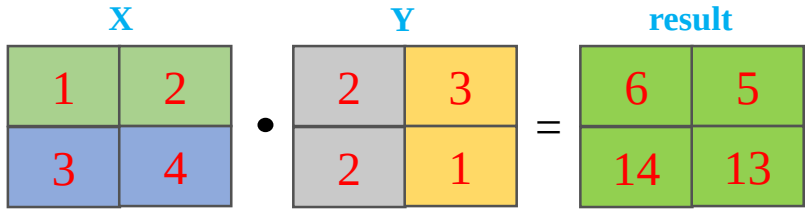

```python
import tensorflow as tf
import numpy as np

matrix = tf.convert_to_tensor([[1, 2], [3, 4]])
vector = tf.convert_to_tensor([1, 2])

print('matrix shape \n', matrix.shape)
print('vector shape \n', vector.shape)

vector = tf.reshape(vector, (2, 1))
print('vector reshape \n', vector.shape)

result1 = tf.matmul(matrix, vector)
print(result1)

result2 = tf.matmul(tf.transpose(vector), matrix)
print(result2)
```

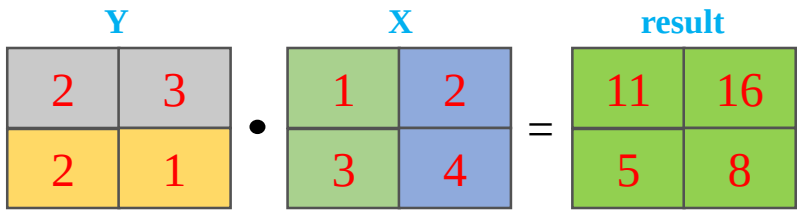
```
matrix shape
 (2, 2)
vector shape
 (2,)
vector reshape
 (2, 1)
tf.Tensor(
[[ 5]
 [11]], shape=(2, 1), dtype=int32)
tf.Tensor([[ 7 10]], shape=(1, 2), dtype=int32)
```

Introduction

❖ Tensor

❖ Important functions





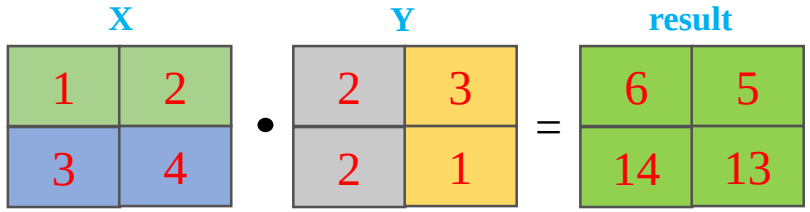

```python
import tensorflow as tf
import numpy as np

matrix1 = tf.convert_to_tensor([[1, 2], [3, 4]])
matrix2 = tf.convert_to_tensor([[2, 3], [2, 1]])

print('matrix1 shape \n', matrix1.shape)
print('matrix2 shape \n', matrix2.shape)

result1 = tf.matmul(matrix1, matrix2)
print(result1)

result2 = tf.matmul(matrix2, matrix1)
print(result2)
```

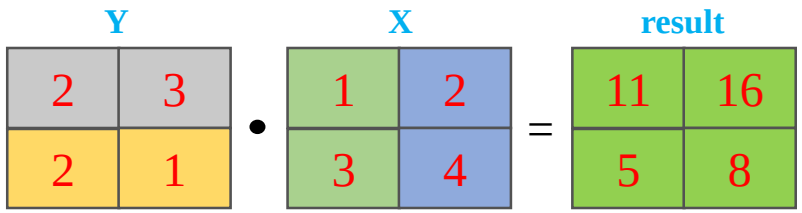
```
matrix1 shape
 (2, 2)
matrix2 shape
 (2, 2)
tf.Tensor(
[[ 6  5]
 [14 13]], shape=(2, 2), dtype=int32)
tf.Tensor(
[[11 16]
 [ 5  8]], shape=(2, 2), dtype=int32)
```

Introduction

❖ Tensor

❖ Important functions






```python
import tensorflow as tf
import numpy as np

matrix1 = tf.convert_to_tensor([[1, 2], [3, 4]])
matrix2 = tf.convert_to_tensor([[2, 3], [2, 1]])

print('matrix1 shape \n', matrix1.shape)
print('matrix2 shape \n', matrix2.shape)

result1 = matrix1@matrix2
print(result1)

result2 = matrix2@matrix1
print(result2)
```

```
matrix1 shape
 (2, 2)
matrix2 shape
 (2, 2)
tf.Tensor(
[[ 6  5]
 [14 13]], shape=(2, 2), dtype=int32)
tf.Tensor(
[[11 16]
 [ 5  8]], shape=(2, 2), dtype=int32)
```

Introduction

❖ **Variable: Represent a tensor whose values can be changed**

```
# variable

var = tf.Variable([1.0, 2.0])
print(var.numpy())

var.assign([7, 9])
print(var.numpy())
```

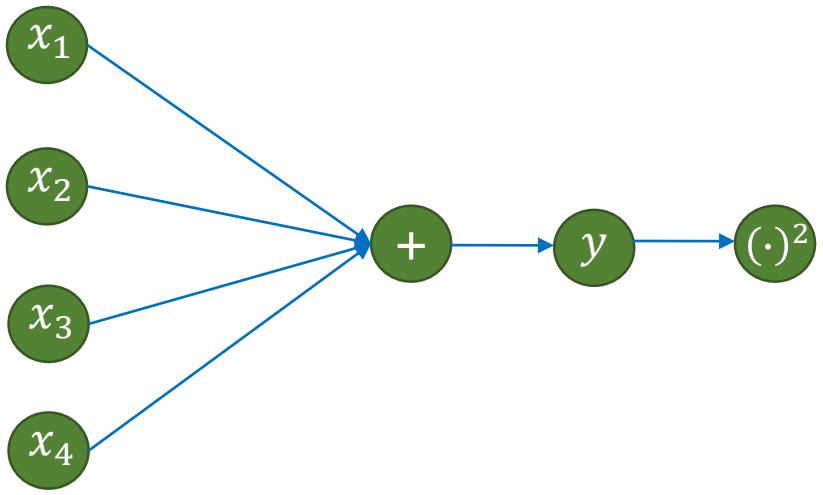
```
[1. 2.]
[7. 9.]
```

```
# variables

var = tf.Variable([1.0, 2.0])

var.assign_add([5, 3])
print(var.numpy())

var.assign_sub([2, 3])
print(var.numpy())
```

```
[6. 5.]
[4. 2.]
```

Introduction

❖ Gradient computation




```
tf.Tensor(
[[1. 1.]
 [1. 1.]], shape=(2, 2), dtype=float32)
tf.Tensor(
[[8. 8.]
 [8. 8.]], shape=(2, 2), dtype=float32)
```

```python
import tensorflow as tf

# data
x = tf.ones((2, 2))

with tf.GradientTape() as t:
    t.watch(x)
    y = tf.reduce_sum(x)
    z = tf.multiply(y, y)

# compute derivative
dz_dx = t.gradient(z, x)

# print out
print(x)
print(dz_dx)
```

Introduction

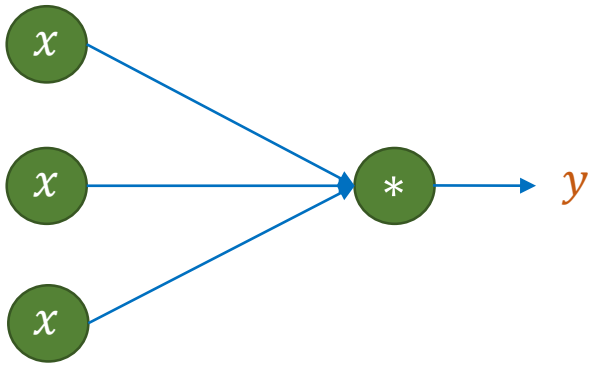
❖ Gradient computation


```
tf.Tensor(108.0, shape=(), dtype=float32)
tf.Tensor(6.0, shape=(), dtype=float32)
```

```python
import tensorflow as tf

# data
x = tf.constant(3.0)

# trace
with tf.GradientTape(persistent=True) as t:
    t.watch(x)
    y = x * x
    z = y * y

# compute derivative
dz_dx = t.gradient(z, x)
dy_dx = t.gradient(y, x)

# print out
print(dz_dx)
print(dy_dx)
```

Introduction

❖ Gradient computation




```
tf.Tensor(3.0, shape=(), dtype=float32)
tf.Tensor(6.0, shape=(), dtype=float32)
```

```python
# data
x = tf.Variable(1.0)

# trace
with tf.GradientTape() as t1:
    with tf.GradientTape() as t2:
        y = x*x*x

    # compute derivative using t2
    dy_dx = t2.gradient(y, x)

# compute derivative using t1
d2y_dx2 = t1.gradient(dy_dx, x)

# print out
print(dy_dx)
print(d2y_dx2)
```

Introduction

❖ Gradient computation

$$g(x) = -3x + 4$$

$$h(g) = 2g + 1$$

```python
import tensorflow as tf

# variable
x = tf.ones((1))*2

# trace
with tf.GradientTape() as tape:
    tape.watch(x)

    # construct functions
    g_x = -3*x + 4
    h_g =  2*g_x + 1

# compute derivative
dh_dx = tape.gradient(h_g, x)

# print out
print(dh_dx)
```

```
tf.Tensor([-6.], shape=(1,), dtype=float32)
```

Introduction

❖ Gradient computation

$$g(x) = x^2 + 1$$

$$h(g) = e^g$$

```
import tensorflow as tf

# variable
x = tf.ones((1))

# trace
with tf.GradientTape() as tape:
    tape.watch(x)

    # construct functions
    g_x = x*x + 1
    h_g = tf.exp(g_x)

# compute derivative
dh_dx = tape.gradient(h_g, x)

# print out
print(dh_dx)
```

```
tf.Tensor([14.778112], shape=(1,), dtype=float32)
```

Introduction

❖ Gradient computation

$$g(x) = cos(x^2 e^x + 2x)$$

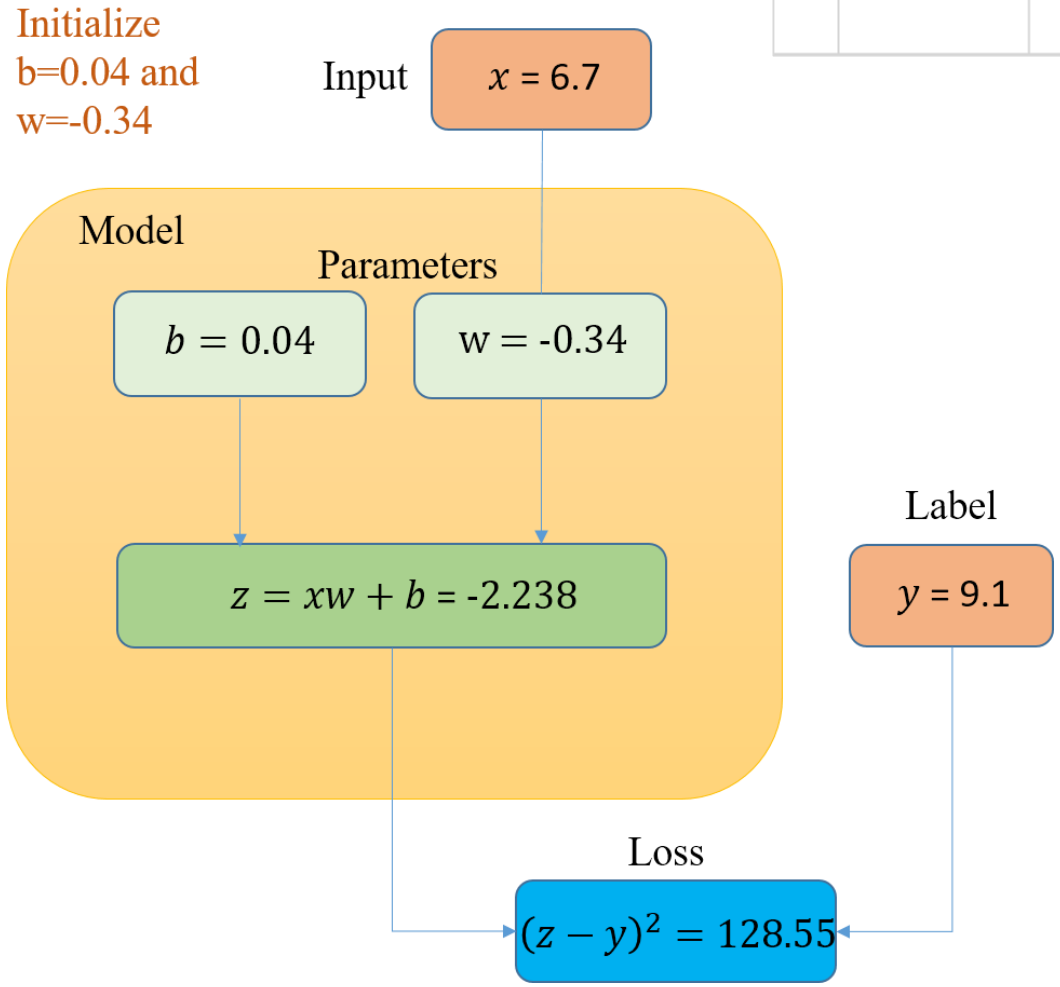
$$h(g) = e^g sin(\sqrt{g})$$

```python
import tensorflow as tf

# variable
x = tf.ones((1))

# trace
with tf.GradientTape() as tape:
    tape.watch(x)

    # construct functions
    g_x = tf.math.cos(x*x*tf.exp(x) + 2*x)
    h_g = tf.exp(g_x)*tf.math.sin(tf.math.sqrt(g_x))

# compute derivative
dh_dx = tape.gradient(h_g, x)

# print out
print(dh_dx)
```

```
tf.Tensor([67.120346], shape=(1,), dtype=float32)
```

Gradient computation

area	price
6.7	9.1
4.6	5.9
3.5	4.6
5.5	6.7

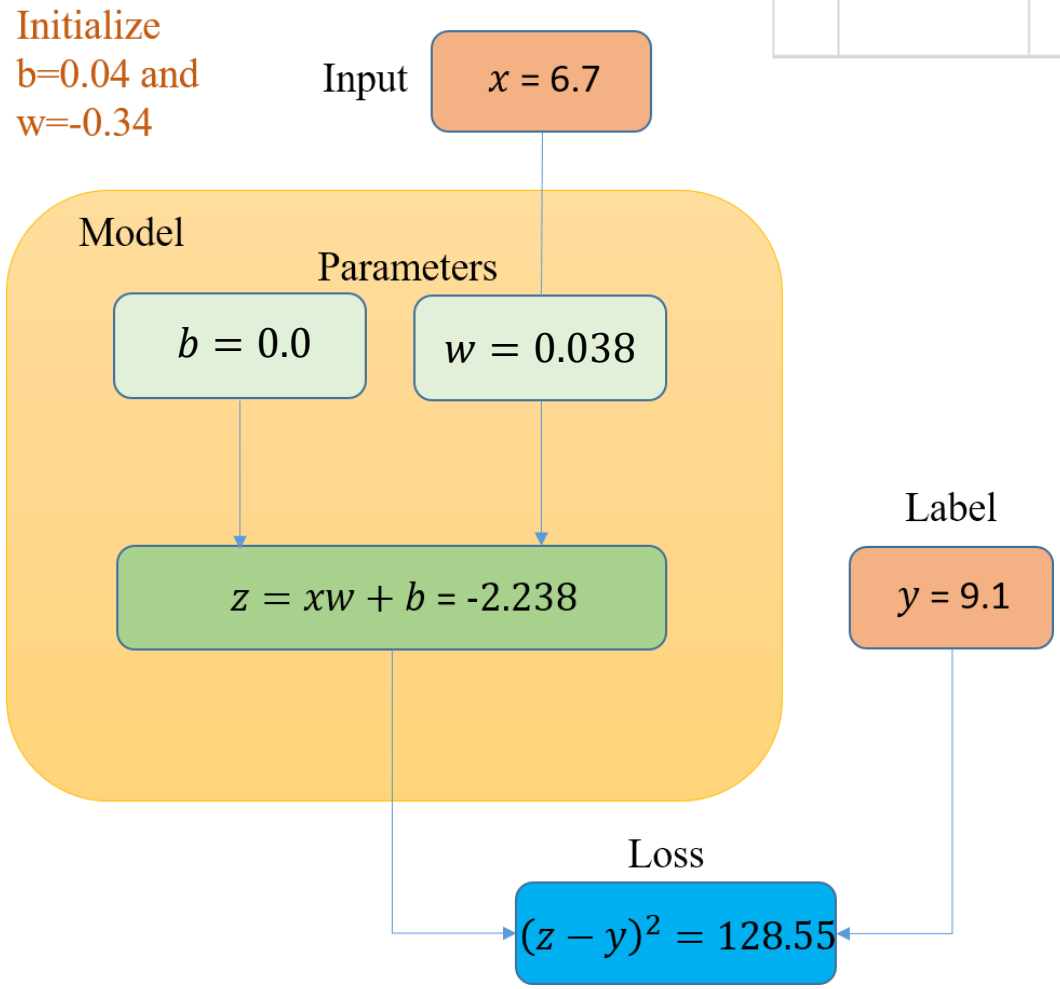



```python
import tensorflow as tf

w = tf.Variable([-0.34])
b = tf.Variable([0.04])

x = [6.7, 4.6, 3.5, 5.5]
x = tf.convert_to_tensor(x)

print('b: \n', b.numpy())
print('w: \n', w.numpy())
print('x: \n', x.numpy())

with tf.GradientTape(persistent=True) as tape:
    y = tf.math.multiply(x, w) + b
    print('y: \n', y.numpy())

    loss = tf.reduce_mean(y**2)
    print('loss: \n', loss.numpy())
```

```
b:
 [0.04]
w:
 [-0.34]
x:
 [6.7 4.6 3.5 5.5]
y:
 [-2.238     -1.524     -1.1500001 -1.83      ]
loss:
 3.000655
```

Gradient computation

area	price
6.7	9.1
4.6	5.9
3.5	4.6
5.5	6.7




```python
import tensorflow as tf

w = tf.Variable(tf.random.normal((1,)))
b = tf.Variable(tf.zeros(1, dtype=tf.float32))

x = [6.7, 4.6, 3.5, 5.5]
x = tf.convert_to_tensor(x)

print('b: \n', b.numpy())
print('w: \n', w.numpy())
print('x: \n', x.numpy())

with tf.GradientTape(persistent=True) as tape:
    y = tf.math.multiply(x, w) + b
    print('y: \n', y.numpy())

    loss = tf.reduce_mean(y**2)
    print('loss: \n', loss.numpy())
```

```
b:
 [0.]
w:
 [0.03887156]
x:
 [6.7 4.6 3.5 5.5]
y:
 [0.2604395  0.1788092  0.13605048 0.2137936 ]
loss:
 0.04100472
```

Introduction

❖ Gradient computation

area	price
6.7	9.1
4.6	5.9
3.5	4.6
5.5	6.7

```
theta:
 [[ 0.04]
 [-0.34]]
x:
 [[1.  6.7]
 [1.  4.6]
 [1.  3.5]
 [1.  5.5]]
y:
 [[-2.238    ]
 [-1.524    ]
 [-1.1500001]
 [-1.83     ]]
loss:
 3.000655
```

```python
import tensorflow as tf

theta = tf.Variable([[0.04], [-0.34]])

x = [[1, 6.7],
     [1, 4.6],
     [1, 3.5],
     [1, 5.5]];
x = tf.convert_to_tensor(x)

print('theta: \n', theta.numpy())
print('x: \n', x.numpy())

with tf.GradientTape(persistent=True) as tape:
    y = x@theta
    print('y: \n', y.numpy())

    loss = tf.reduce_mean(y**2)
    print('loss: \n', loss.numpy())
```

Introduction

❖ Gradient computation

area	price
6.7	9.1
4.6	5.9
3.5	4.6
5.5	6.7

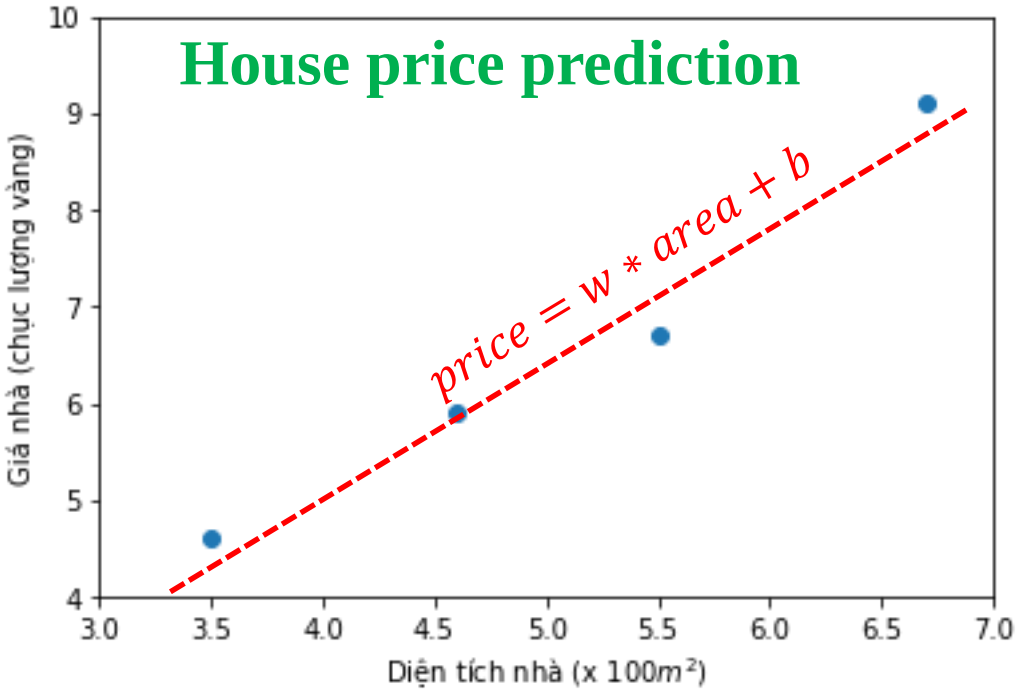
```
theta:
 [[0.28044993]
 [1.5637906 ]]
x:
 [[1.  6.7]
 [1.  4.6]
 [1.  3.5]
 [1.  5.5]]
y:
 [[10.757846 ]
 [ 7.4738865]
 [ 5.753717 ]
 [ 8.881298 ]]
loss:
 70.893234
```

```python
import tensorflow as tf

theta = tf.Variable(tf.random.normal((2,1)))

x = [[1, 6.7],
     [1, 4.6],
     [1, 3.5],
     [1, 5.5]];
x = tf.convert_to_tensor(x)

print('theta: \n', theta.numpy())
print('x: \n', x.numpy())

with tf.GradientTape(persistent=True) as tape:
    y = x@theta
    print('y: \n', y.numpy())

    loss = tf.reduce_mean(y**2)
    print('loss: \n', loss.numpy())
```

Linear Regression

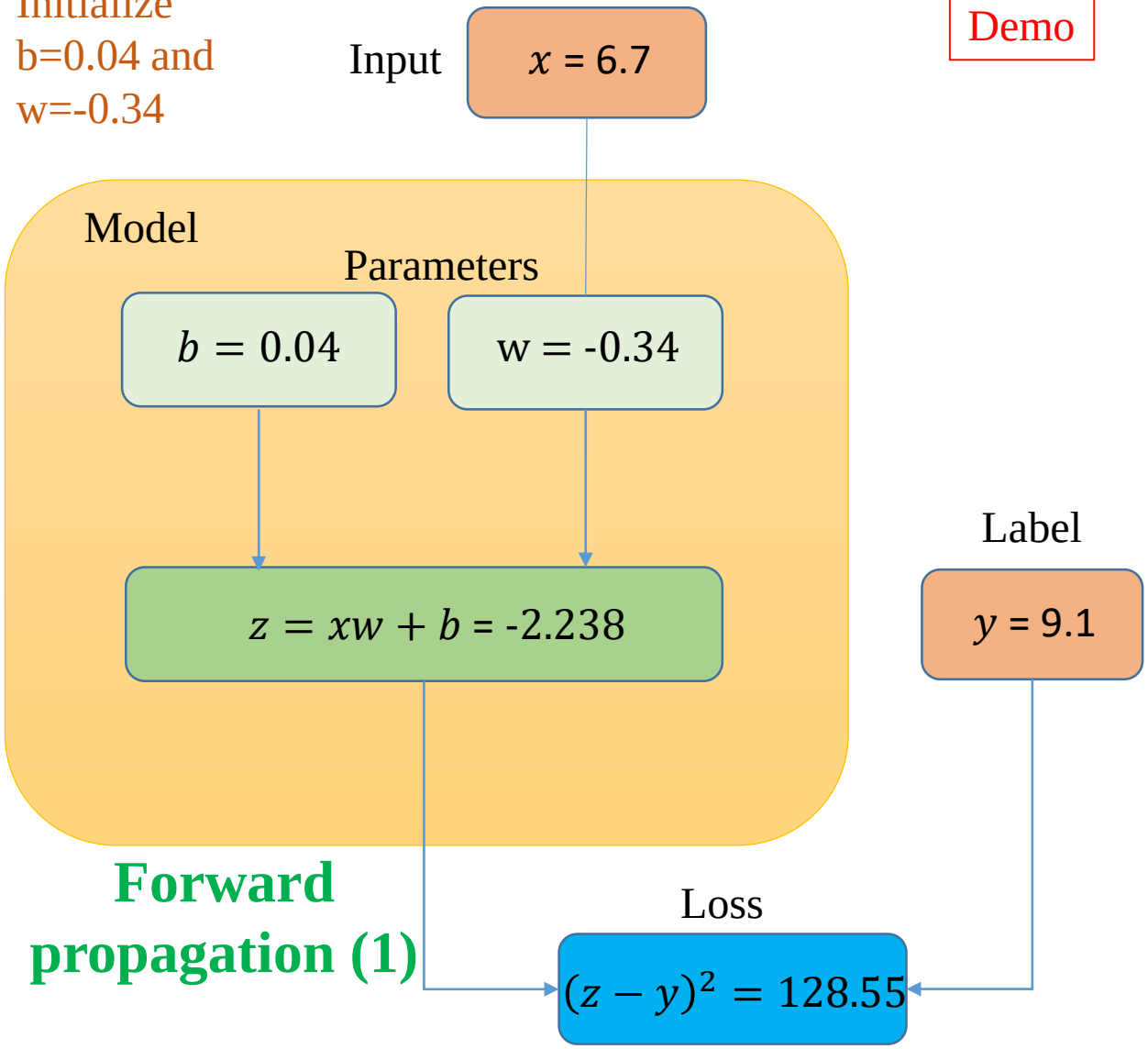
Given sample data

area	price
6.7	9.1
4.6	5.9
3.5	4.6
5.5	6.7



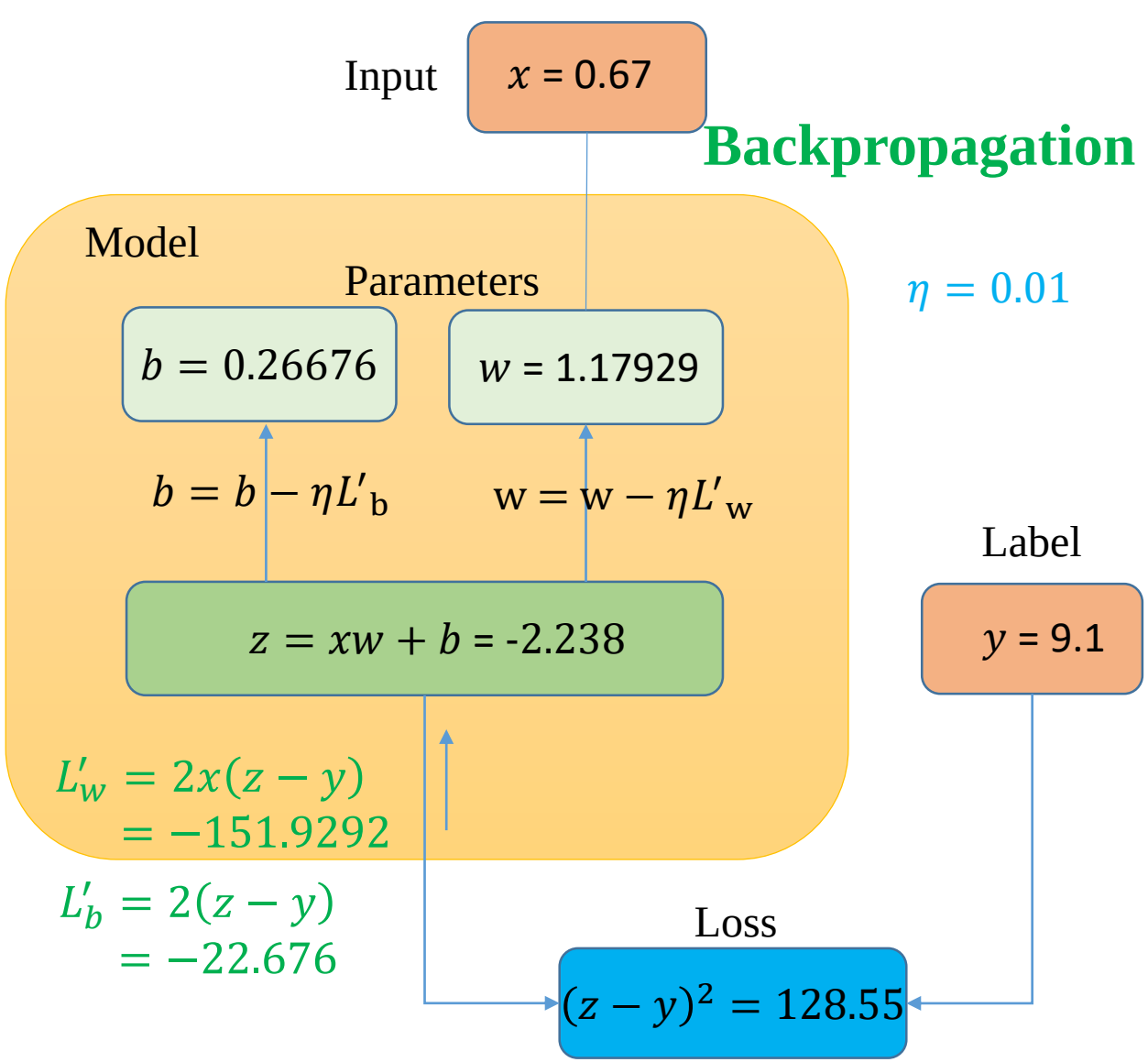
Initialize b=0.04 and w=-0.34

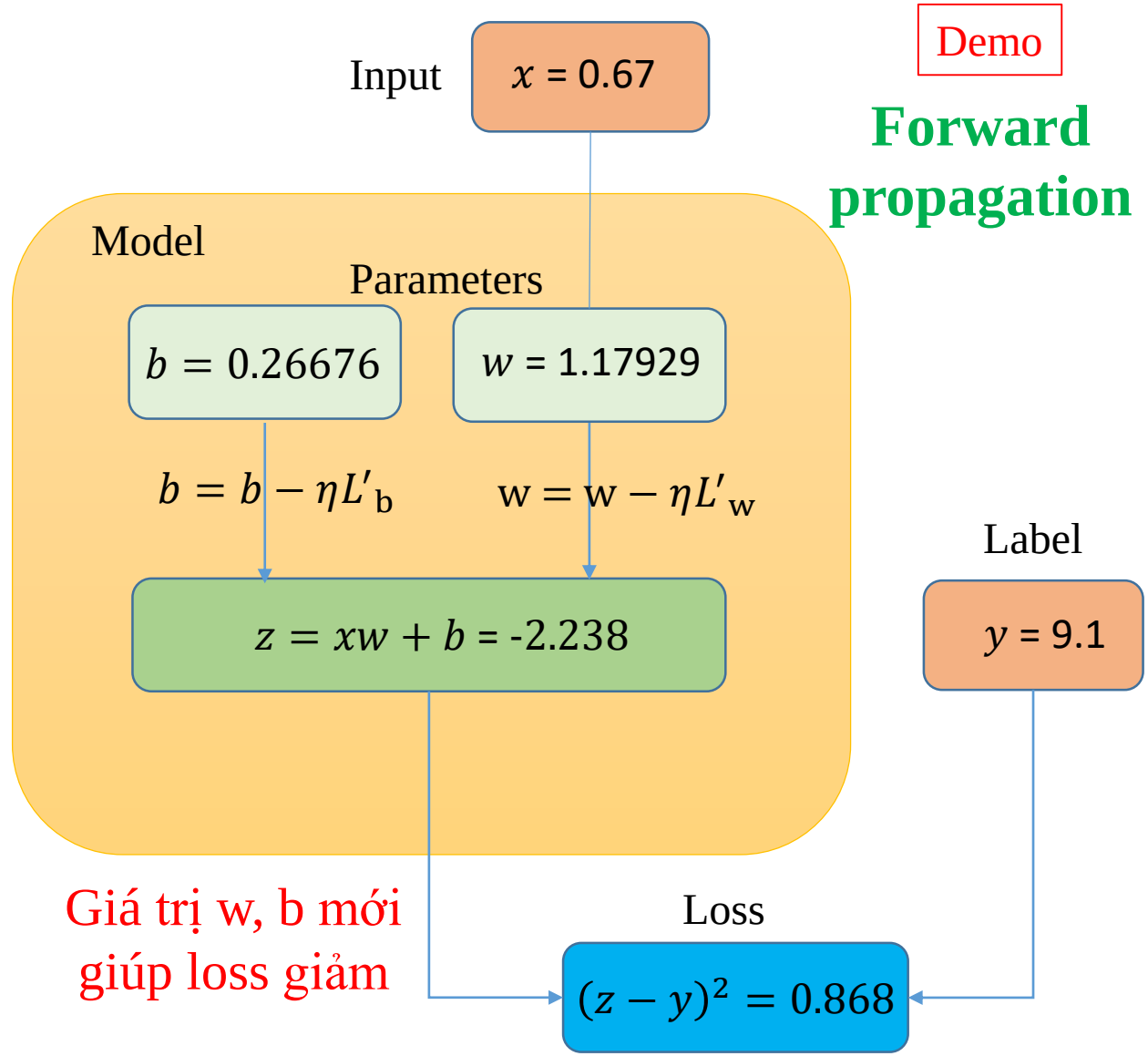
Demo



Forward propagation (1)

Linear Regression





Introduction

❖ Gradient computation

```
theta:
 [[ 0.04]
 [-0.34]]
x:
 [[1.  6.7]]
y:
 [[-2.238]]
loss:
 128.55025
```

```
tf.Tensor(
[[ -22.676 ]
 [-151.9292]], shape=(2, 1), dtype=float32)
```

```python
import tensorflow as tf

theta = tf.Variable([[0.04], [-0.34]])

x = [[1, 6.7]];
x = tf.convert_to_tensor(x)

y = [9.1];

print('theta: \n', theta.numpy())
print('x: \n', x.numpy())

with tf.GradientTape(persistent=True) as tape:
    y_hat = x@theta
    print('y: \n', y_hat.numpy())

    loss = tf.reduce_mean((y_hat-y)**2)
    print('loss: \n', loss.numpy())

    dtheta = tape.gradient(loss, theta)
    print(dtheta)
```

Loss Functions

❖ Cross-entropy

$$\text{Loss} = \sum_{i} -y_i \log(\hat{y}_i)$$

```python
import tensorflow as tf

y_true = [[0, 1, 0], [0, 0, 1]]
y_pred = [[0.05, 0.95, 0], [0.1, 0.8, 0.1]]
y_pred = tf.convert_to_tensor(y_pred)

# CategoricalCrossentropy
ce_function = tf.keras.losses.CategoricalCrossentropy()
loss = ce_function(y_true, y_pred)

print(loss.numpy())
```

```
1.1769392
```

Loss Functions

❖ Cross-entropy

$$\text{Loss} = \sum_{i} -y_i \log(\hat{y}_i)$$

```python
import tensorflow as tf

y_true = [1, 2]
y_pred = [[0.05, 0.95, 0], [0.1, 0.8, 0.1]]
y_pred = tf.convert_to_tensor(y_pred)

# SparseCategoricalCrossentropy
sce_function = tf.keras.losses.SparseCategoricalCrossentropy()
loss = sce_function(y_true, y_pred)
print(loss.numpy())
```

```
1.1769392
```

Keras

- Run on top of Tensorflow
- Integrated into Tensorflow

Package Declaration

```
1  import tensorflow as tf
2  import tensorflow.keras as keras
3
4  print("TensorFlow version: ", tf.__version__)
5  print("Keras version: ", keras.__version__)
```

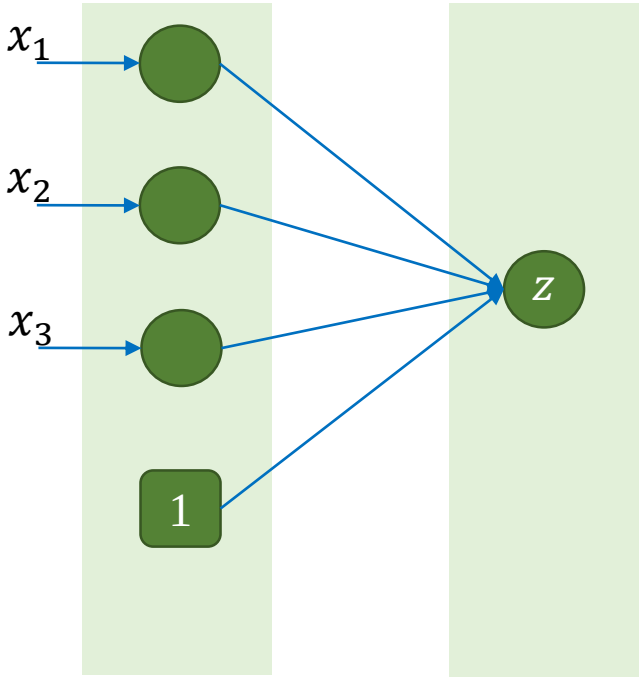
```
TensorFlow version:  2.7.0
Keras version:  2.7.0
```

Original author(s)	François Chollet
Developer(s)	various
Initial release	27 March 2015; 6 years ago
Stable release	2.4.0[1] / 17 June 2020; 17 months ago
Repository	github.com/keras-team/keras
Written in	Python
Platform	Cross-platform
Type	Neural networks
License	MIT
Website	keras.io

https://en.wikipedia.org/wiki/Keras

Keras

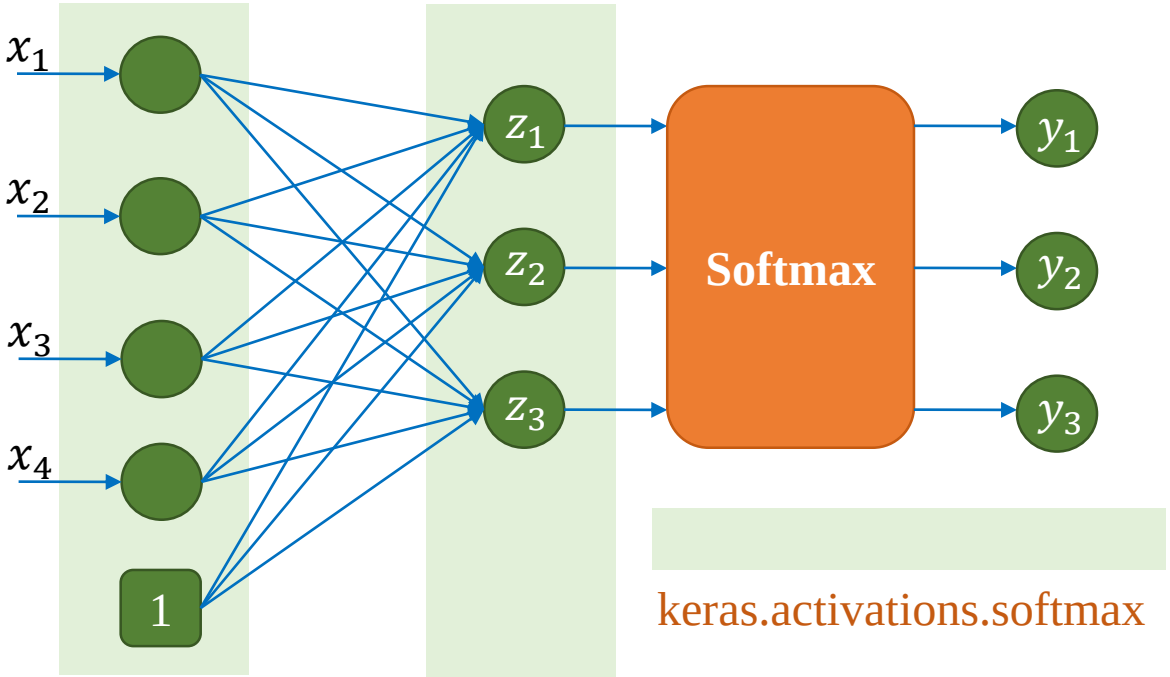
Model



keras.Input(shape=(3,))

keras.layers.Dense(units=1)

Model



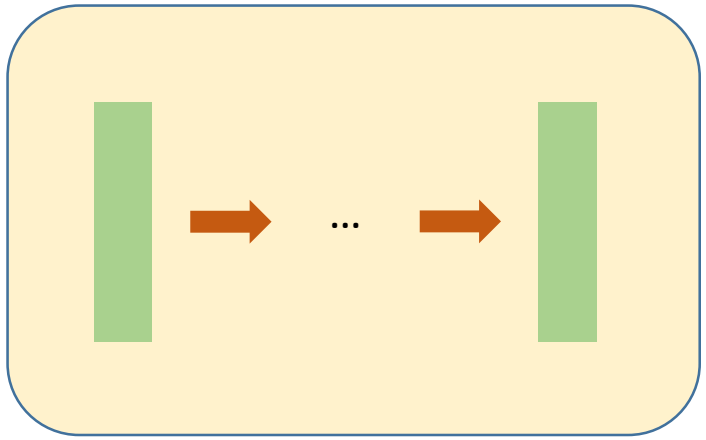
keras.Input(shape=(4,))

keras.activations.softmax

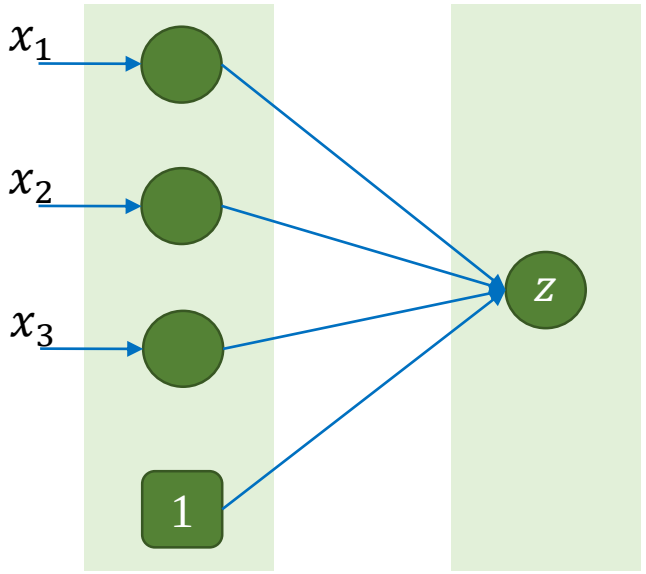
keras.layers.Dense(units=3)

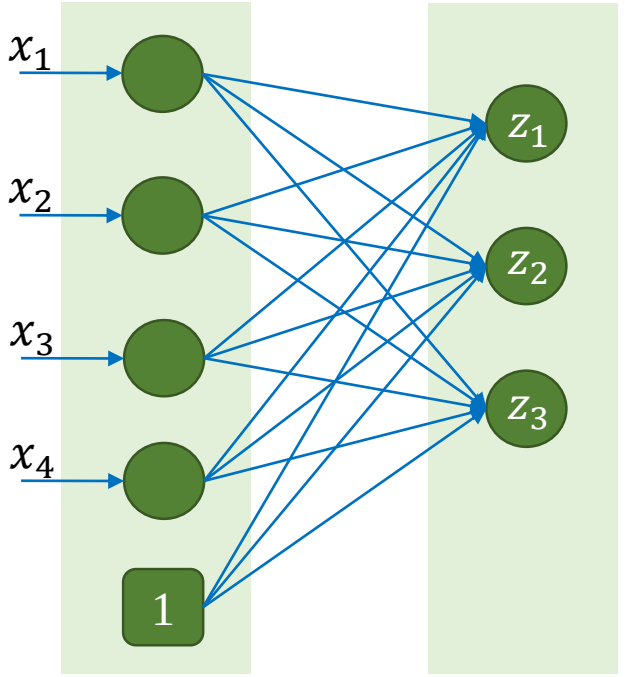
Keras

Model



keras.Sequential()



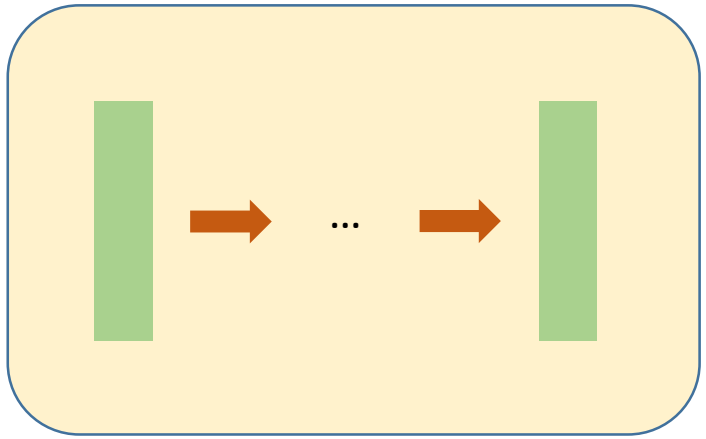



```python
import numpy as np
import tensorflow as tf
import tensorflow.keras as keras

# create model
model = keras.Sequential()
model.add(keras.Input(shape=(3,)))
model.add(keras.layers.Dense(1))
```

```python
import numpy as np
import tensorflow as tf
import tensorflow.keras as keras

# create model
model = keras.Sequential()
model.add(keras.Input(shape=(4,)))
model.add(keras.layers.Dense(3))
```

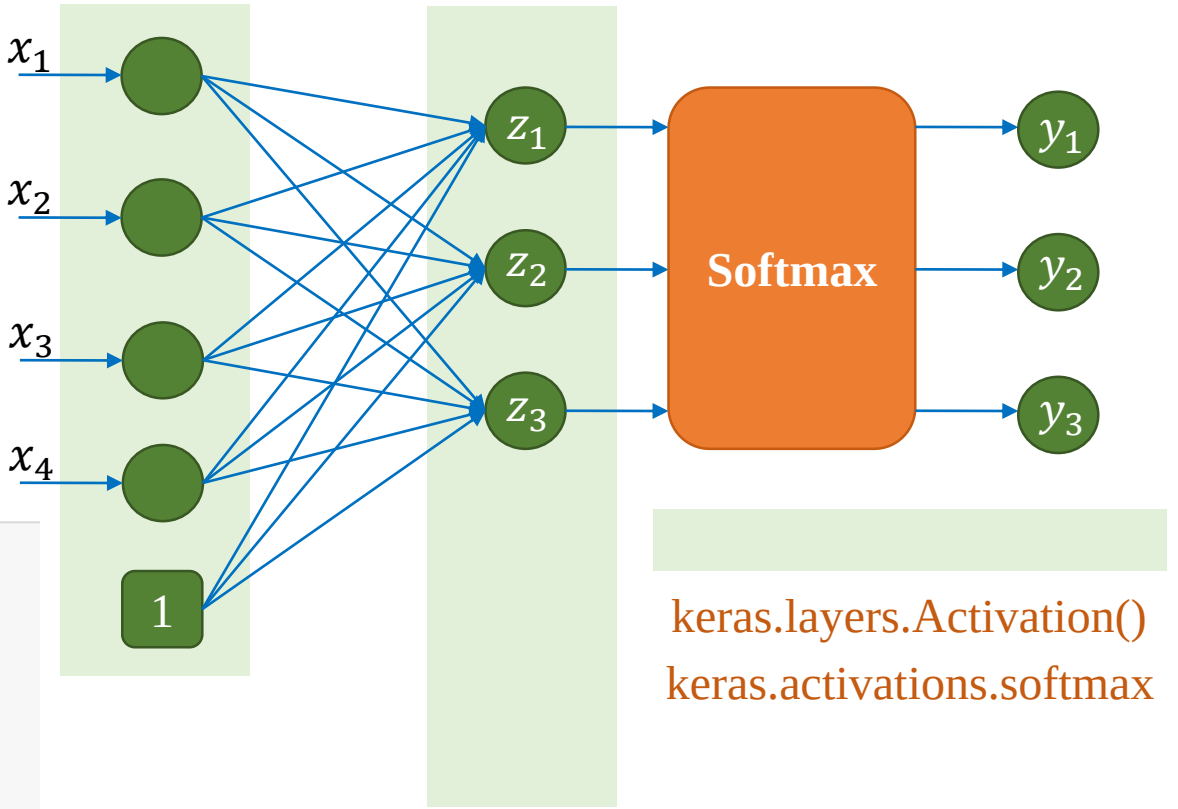
Keras

Model



keras.Sequential()

keras.Input(shape=(4,))



keras.layers.Activation()
keras.activations.softmax

keras.layers.Dense(units=3)

```
import numpy as np
import tensorflow as tf
import tensorflow.keras as keras

# create model
model = keras.Sequential()
model.add(keras.Input(shape=(4,)))
model.add(keras.layers.Dense(3))
model.add(keras.layers.Activation(keras.activations.softmax))
```

Outline

- Introduction to Tensorflow
- Introduction to Tensforflow.Keras
- Model Construction
- Model Training and Inference
- Applying Softmax for Image Data

Model Construction

❖ Linear regression

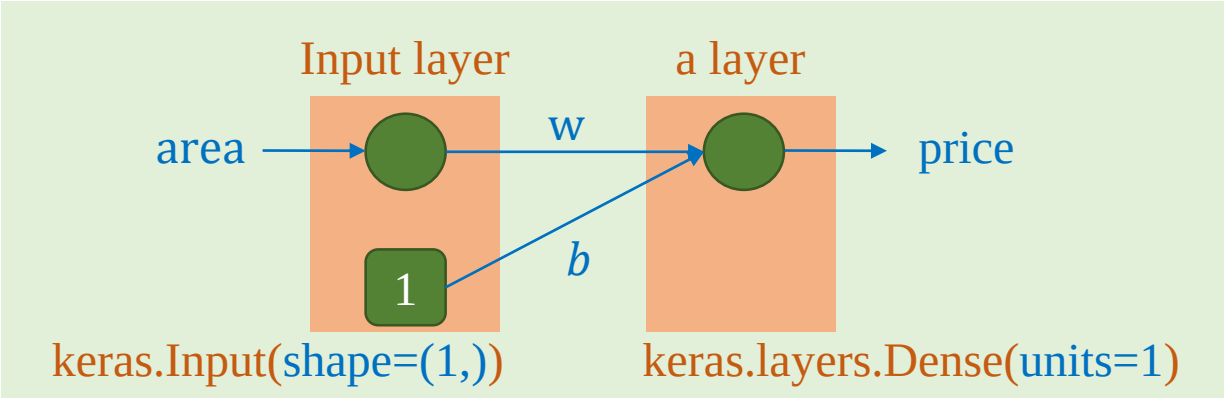
Feature	Label
area	price
6.7	9.1
4.6	5.9
3.5	4.6
5.5	6.7

House price data

Model

$$\text{price} = w * area + b$$

$$y = wx + b$$




```python
import tensorflow as tf
import tensorflow.keras as keras

# create model
model = keras.Sequential()
model.add(keras.Input(shape=(1,)))
model.add(keras.layers.Dense(1))

model.summary()
```

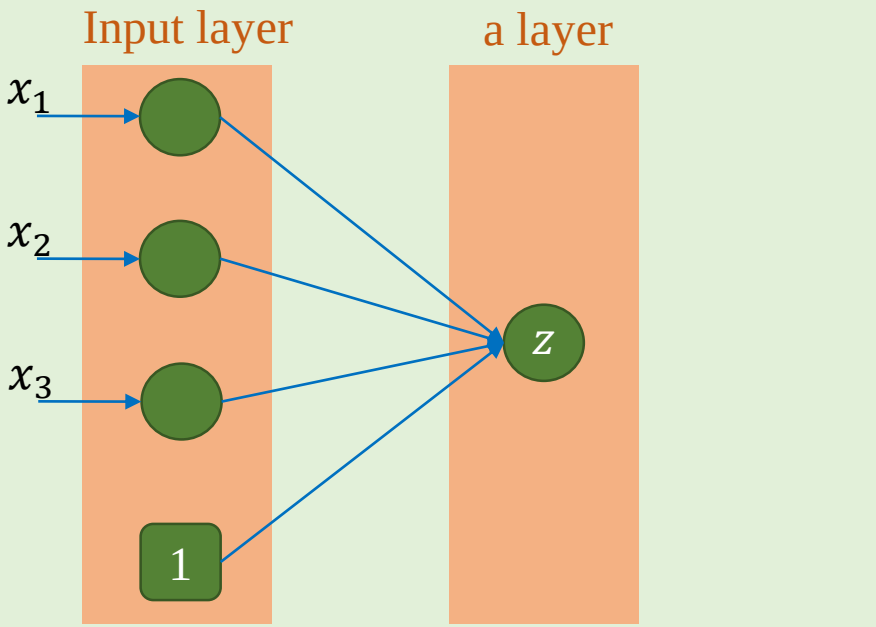
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 1)                 2
=================================================================
Total params: 2
Trainable params: 2
Non-trainable params: 0
_________________________________________________________________
```

Model Construction

❖ Linear regression

Features			Label
TV	Radio	Newspaper	Sales
230.1	37.8	69.2	22.1
44.5	39.3	45.1	10.4
17.2	45.9	69.3	12
151.5	41.3	58.5	16.5
180.8	10.8	58.4	17.9

Advertising-based sale data



Model

$$\text{Sale} = w_1 * TV + w_2 * Radio + w_3 * Newspaper + b$$

$$y = w_1 x_1 + w_2 x_2 + w_3 x_3 + b$$

```
1  import tensorflow as tf
2  import tensorflow.keras as keras
3
4  # create model
5  model = keras.Sequential()
6  model.add(keras.Input(shape=(3,)))
7  model.add(keras.layers.Dense(1))
8
9  model.summary()
```

Model Construction

❖ Linear regression

Boston House Price Data

Features													Label
crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	black	lstat	medv
0.00632	18	2.31	0	0.538	6.575	65.2	4.09	1	296	15.3	396.9	4.98	24
0.02731	0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.9	9.14	21.6
0.03237	0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33.4
0.06905	0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.9	5.33	36.2
0.08829	12.5	7.87	0	0.524	6.012	66.6	5.5605	5	311	15.2	395.6	12.43	22.9

Model

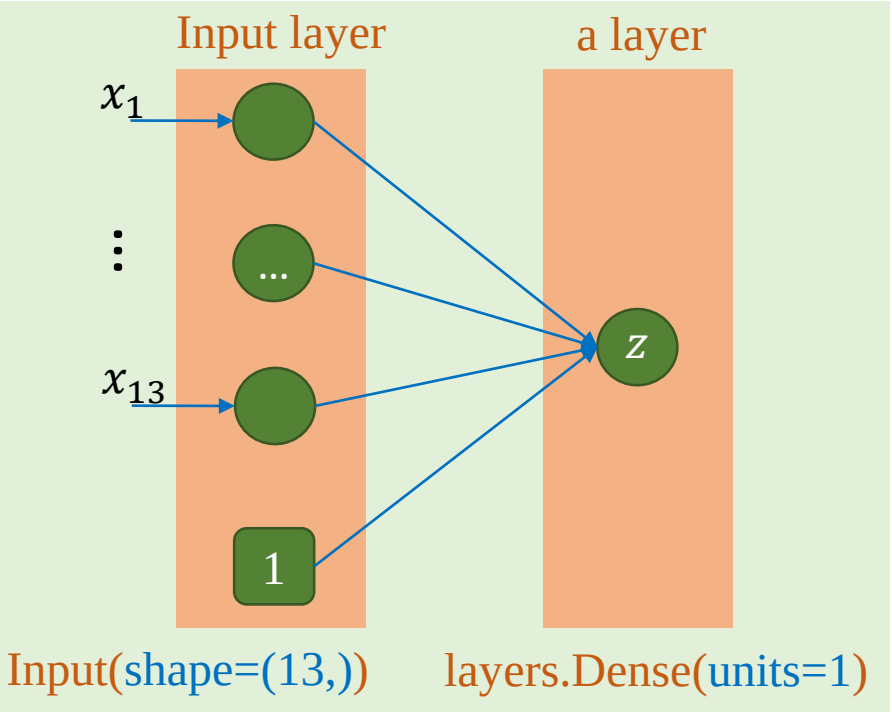
$$\text{medv} = w_1 * x_1 + \cdots + w_{13} * x_{13} + b$$

Model Construction

❖ Linear regression

Model

$$\text{medv} = w_1 * x_1 + \cdots + w_{13} * x_{13} + b$$




```python
import tensorflow as tf
import tensorflow.keras as keras

# create model
model = keras.Sequential()
model.add(keras.Input(shape=(13,)))
model.add(keras.layers.Dense(1))

model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 1)                 14
=================================================================
Total params: 14
Trainable params: 14
Non-trainable params: 0
_________________________________________________________________
```

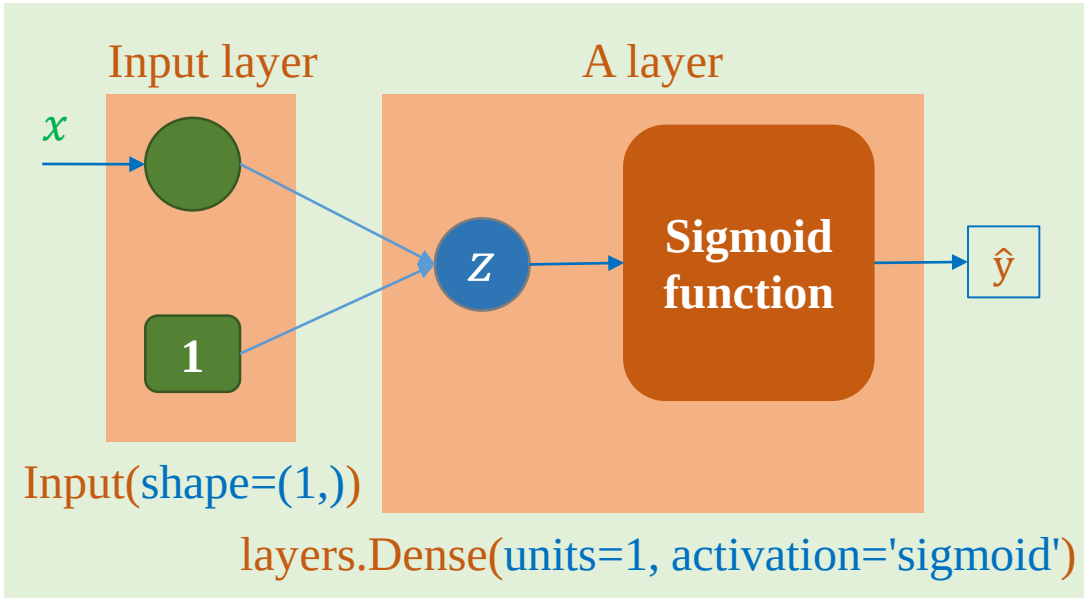
Model Construction

Feature	Label
Petal_Length	Category
1.4	0
1	0
1.5	0
3	1
3.8	1
4.1	1

Model

$$z = wx + b$$

$$\hat{y} = \frac{1}{1 + e^{-z}}$$



❖ Logistic regression

```
1  import tensorflow as tf
2  import tensorflow.keras as keras
3
4  # create model
5  model = keras.Sequential()
6  model.add(keras.Input(shape=(1,)))
7  model.add(keras.layers.Dense(1, activation='sigmoid'))
8
9  model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 1)                 2
=================================================================
Total params: 2
Trainable params: 2
Non-trainable params: 0
_________________________________________________________________
```

Model Construction

❖ Logistic regression

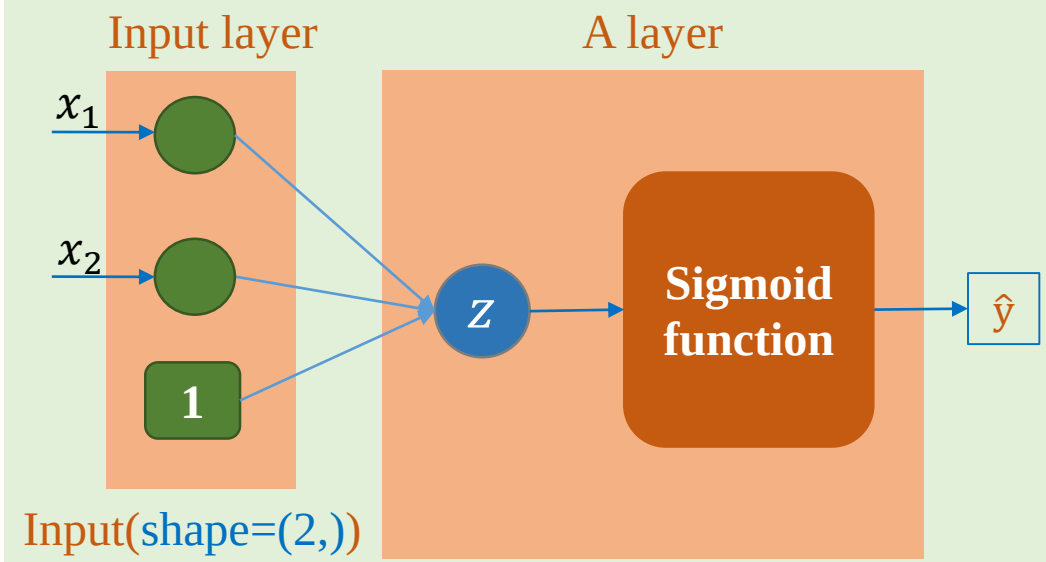
Feature **Label**

Petal_Length	Petal_Width	Label
1.5	0.2	0
1.4	0.2	0
1.6	0.2	0
4.7	1.6	1
3.3	1.1	1
4.6	1.3	1

Model

$$z = \theta^T x$$

$$\hat{y} = \frac{1}{1 + e^{-z}}$$




```python
import tensorflow as tf
import tensorflow.keras as keras

# create model
model = keras.Sequential()
model.add(keras.Input(shape=(2,)))
model.add(keras.layers.Dense(1, activation='sigmoid'))

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 1)                 3
=================================================================
Total params: 3
Trainable params: 3
Non-trainable params: 0
_________________________________________________________________
```

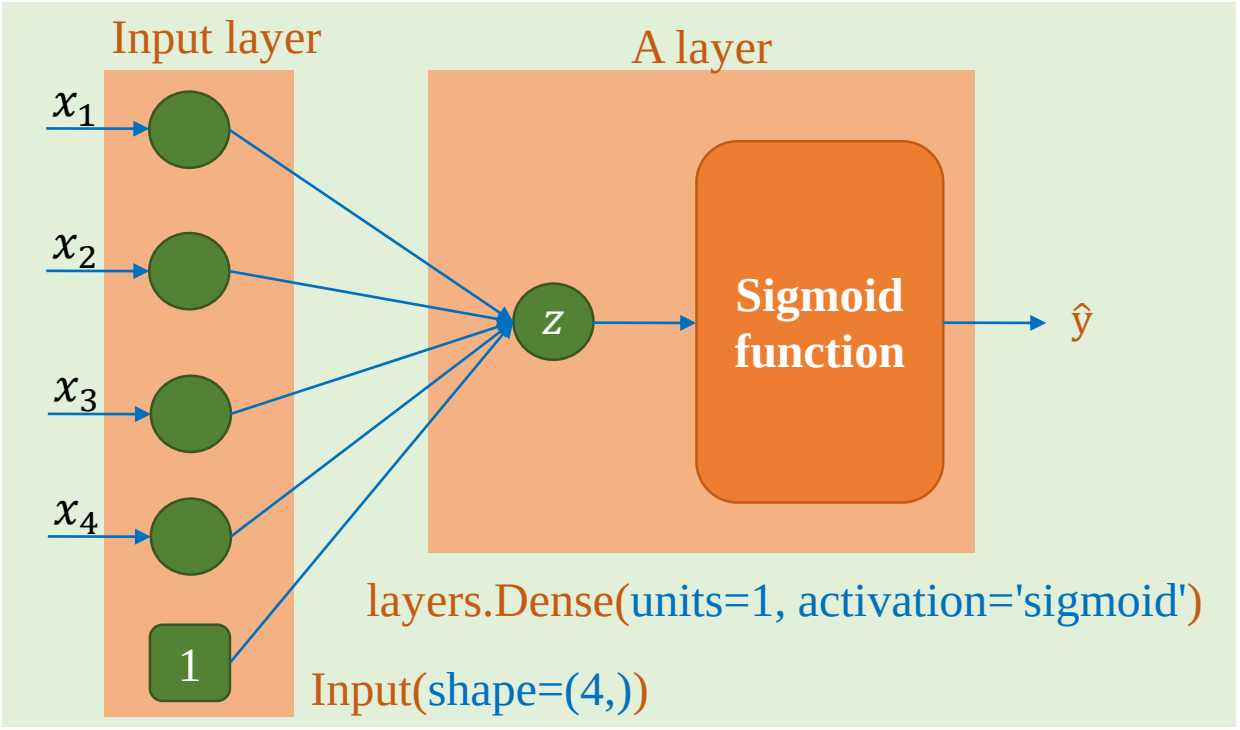
Model Construction

Feature				Label
Sepal_Length	Sepal_Width	Petal_Length	Petal_Width	Label
5.2	3.5	1.5	0.2	0
5.2	3.4	1.4	0.2	0
4.7	3.2	1.6	0.2	0
6.3	3.3	4.7	1.6	1
4.9	2.4	3.3	1.1	1
6.6	2.9	4.6	1.3	1

Model

$$z = \theta^T x$$

$$\hat{y} = \frac{1}{1 + e^{-z}}$$



❖ Logistic regression

```python
import tensorflow as tf
import tensorflow.keras as keras

# create model
model = keras.Sequential()
model.add(keras.Input(shape=(4,)))
model.add(keras.layers.Dense(1, activation='sigmoid'))

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Para
=================================================================
dense (Dense)                (None, 1)                 5
=================================================================
Total params: 5
Trainable params: 5
Non-trainable params: 0
_________________________________________________________________
```

AI VIETNAM
All-in-One Course

Model Construction

Feature	Label
Petal_Length	Label
1.4	1
1.3	1
1.5	1
4.5	2
4.1	2
4.6	2

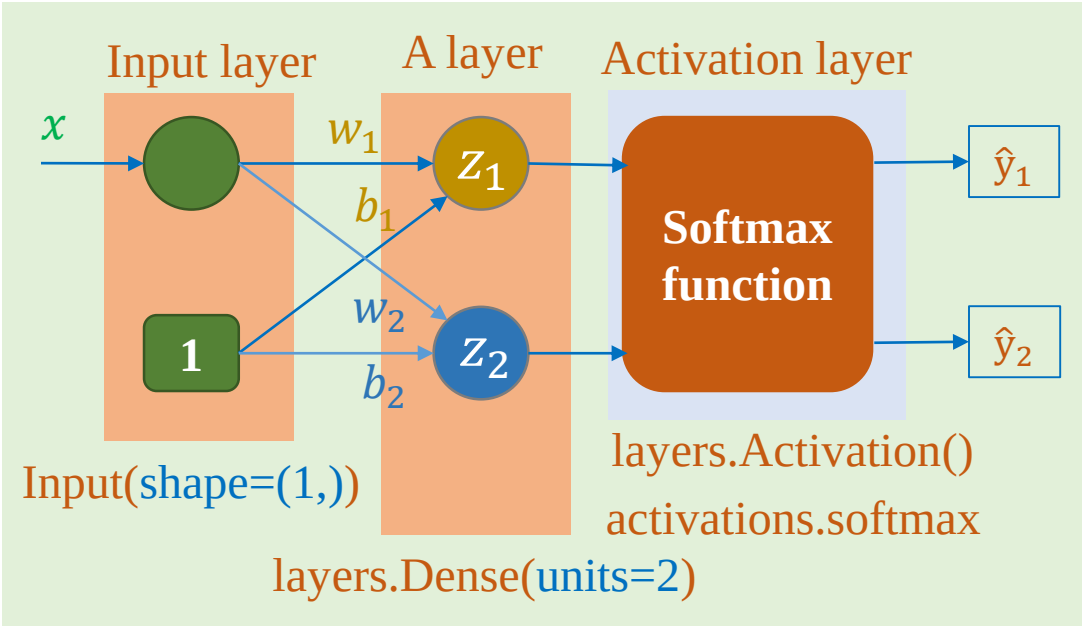
Iris Classification Data

$$z_1 = xw_1 + b_1$$

$$z_2 = xw_2 + b_2$$

$$\hat{y}_1 = \frac{e^{z_1}}{\sum_{j=1}^{2} e^{z_j}}$$

$$\hat{y}_2 = \frac{e^{z_1}}{\sum_{j=1}^{2} e^{z_j}}$$



❖ Softmax regression

```
1  import tensorflow as tf
2  import tensorflow.keras as keras
3
4  # create model
5  model = keras.Sequential()
6  model.add(keras.Input(shape=(1,)))
7  model.add(keras.layers.Dense(2))
8  model.add(keras.layers.Activation(keras.activations.softmax))
9
10 model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 2)                 4
_________________________________________________________________
activation (Activation)      (None, 2)                 0
=================================================================
Total params: 4
Trainable params: 4
Non-trainable params: 0
_________________________________________________________________
```

Model Construction

Feature	Label
Petal_Length	Label
1.4	1
1.3	1
1.5	1
4.5	2
4.1	2
4.6	2

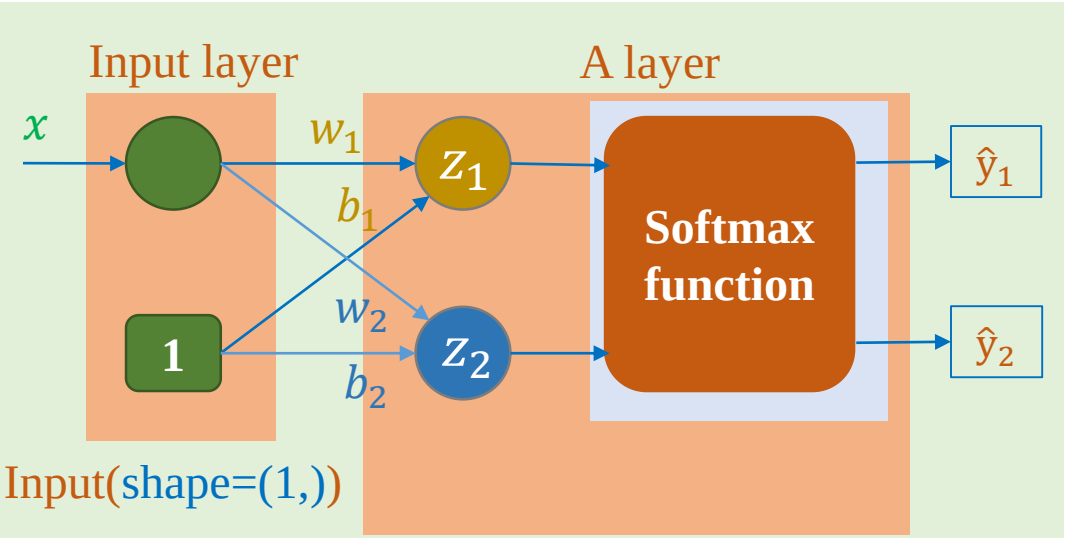
Iris Classification Data

$$z_1 = xw_1 + b_1$$

$$z_2 = xw_2 + b_2$$

$$\hat{y}_1 = \frac{e^{z_1}}{\sum_{j=1}^{2} e^{z_j}}$$

$$\hat{y}_2 = \frac{e^{z_1}}{\sum_{j=1}^{2} e^{z_j}}$$



❖ Softmax regression

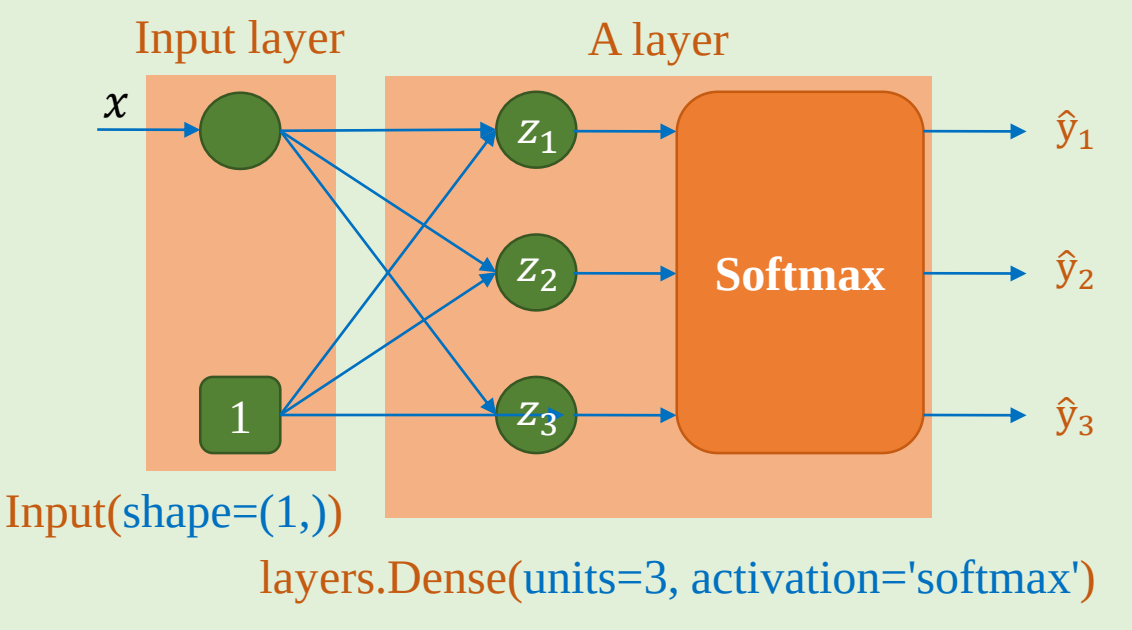
```
1  import tensorflow as tf
2  import tensorflow.keras as keras
3
4  # create model
5  model = keras.Sequential()
6  model.add(keras.Input(shape=(1,)))
7  model.add(keras.layers.Dense(2, activation='softmax'))
8
9  model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 2)                 4
=================================================================
Total params: 4
Trainable params: 4
Non-trainable params: 0
_________________________________________________________________
```

Model Construction

❖ Softmax regression

Petal_Length	Label
1.4	1
1.3	1
1.5	1
4.5	2
4.1	2
4.6	2
5.2	3
5.6	3
5.9	3



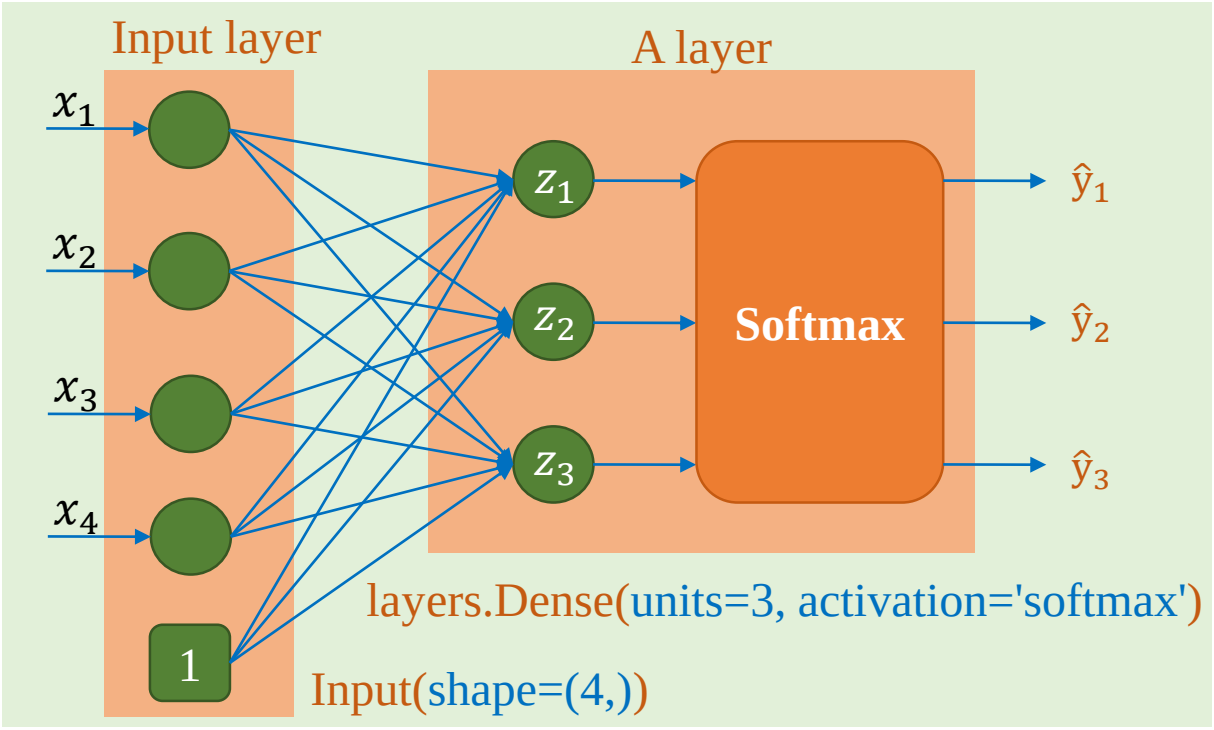

```python
import tensorflow as tf
import tensorflow.keras as keras

# create model
model = keras.Sequential()
model.add(keras.Input(shape=(1,)))
model.add(keras.layers.Dense(3, activation='softmax'))

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 3)                 6
=================================================================
Total params: 6
Trainable params: 6
Non-trainable params: 0
_________________________________________________________________
```

AI VIETNAM
All-in-One Course

Model Construction

Sepal_Length	Sepal_Width	Petal_Length	Petal_Width	Label
5.2	3.5	1.5	0.2	1
5.2	3.4	1.4	0.2	1
4.7	3.2	1.6	0.2	1
6.3	3.3	4.7	1.6	2
4.9	2.4	3.3	1.1	2
6.6	2.9	4.6	1.3	2
6.4	2.8	5.6	2.2	3
6.3	2.8	5.1	1.5	3
6.1	2.6	5.6	1.4	3



❖ Softmax regression

Forward computation

$$z = \theta^T x \qquad \hat{y} = \frac{e^z}{\sum_{i=1}^{k} e^{z_i}}$$

```python
import tensorflow as tf
import tensorflow.keras as keras

# create model
model = keras.Sequential()
model.add(keras.Input(shape=(4,)))
model.add(keras.layers.Dense(3, activation='softmax'))

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 3)                 15
=================================================================
Total params: 15
Trainable params: 15
Non-trainable params: 0
_________________________________________________________________
```

Outline

- Introduction to Tensorflow
- Introduction to Tensforflow.Keras
- Model Construction
- Model Training and Inference
- Applying Softmax for Image Data

Training

❖ Logistic regression

→ Tính output $\hat{y}$

$$z = \theta^T x$$
$$\hat{y} = \sigma(z) = \frac{1}{1 + e^{-z}}$$

→ Tính loss (binary cross-entropy)

$$L(\theta) = \left(-\mathbf{y}^{\mathrm{T}}\log\hat{\mathbf{y}} - (\mathbf{1}-\mathbf{y})^{\mathrm{T}}\log(1-\hat{\mathbf{y}})\right)$$

→ Tính đạo hàm

$$L'_{\theta} = \mathbf{x}^{\mathrm{T}}(\hat{\mathbf{y}} - \boldsymbol{y})$$

→ Cập nhật tham số (Stochastic gradient descent)

$$\boldsymbol{\theta} = \boldsymbol{\theta} - \eta L'_{\theta}$$

Computed automatically

Declare optimizer and loss function

```
model.compile(optimizer='sgd',
              loss='binary_crossentropy')
```

Start training

```
model.fit(x-data, y-data, batch-size, epochs)
```

If batch-size=1 → Stochastic training

If batch-size=N → Batch training

If 1<batch-size<N → Mini-batch training

Training

❖ Softmax regression

→ Tính output $\hat{y}$

$$z = \theta^T x \qquad \hat{y} = \frac{e^z}{\sum_{i=1}^{k} e^{z_i}}$$

→ Tính loss (cross-entropy)

$$L(\theta) = -\sum_{i=1}^{k} y_i \log \hat{y}_i$$

→ Tính đạo hàm

$$\frac{\partial L}{\partial \theta_i} = x(\hat{y}_i - y_i)$$

→ Cập nhật tham số (Stochastic gradient descent)

$$\theta = \theta - \eta L'_\theta$$

Computed automatically

Declare optimizer and loss function

```
model.compile(optimizer='sgd',
              loss='categorical_crossentropy')
```

Start training

```
model.fit(x-data, y-data, batch-size, epochs)
```

If batch-size=1 → Stochastic training

If batch-size=m → Batch training

If 1<batch-size<m → Mini-batch training

AI VIETNAM
All-in-One Course

Training

❖ Linear regression

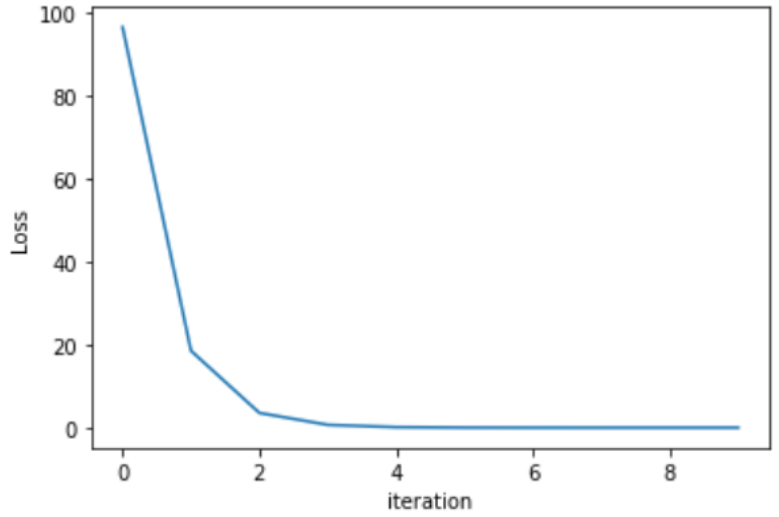
Feature	Label
area	price
6.7	9.1
4.6	5.9
3.5	4.6
5.5	6.7

House price data

Model

$$\text{price} = w * area + b$$

$$y = wx + b$$




```python
import numpy as np
import tensorflow as tf
import tensorflow.keras as keras

batch_size = 4
epochs = 10

# Data Preparation
data = np.genfromtxt('data.csv', delimiter=',')
X = data[:,0:1]
y = data[:,1:]

# create model
model = tf.keras.Sequential(
    [tf.keras.layers.Dense(units=1, input_shape=[1])])

# declare optimization method and loss function
opt = keras.optimizers.SGD(learning_rate=0.01)
model.compile(optimizer=opt, loss='mse')

# training
history = model.fit(X, y, batch_size, epochs)
```

```
Train on 4 samples
Epoch 1/10
4/4 [==============================] - 0s 28ms/sample - loss: 96.6016
Epoch 2/10
4/4 [==============================] - 0s 247us/sample - loss: 18.6414
Epoch 3/10
4/4 [==============================] - 0s 499us/sample - loss: 3.6692
Epoch 4/10
4/4 [==============================] - 0s 245us/sample - loss: 0.7937
Epoch 5/10
4/4 [==============================] - 0s 499us/sample - loss: 0.2415
Epoch 6/10
4/4 [==============================] - 0s 499us/sample - loss: 0.1353
```

Training

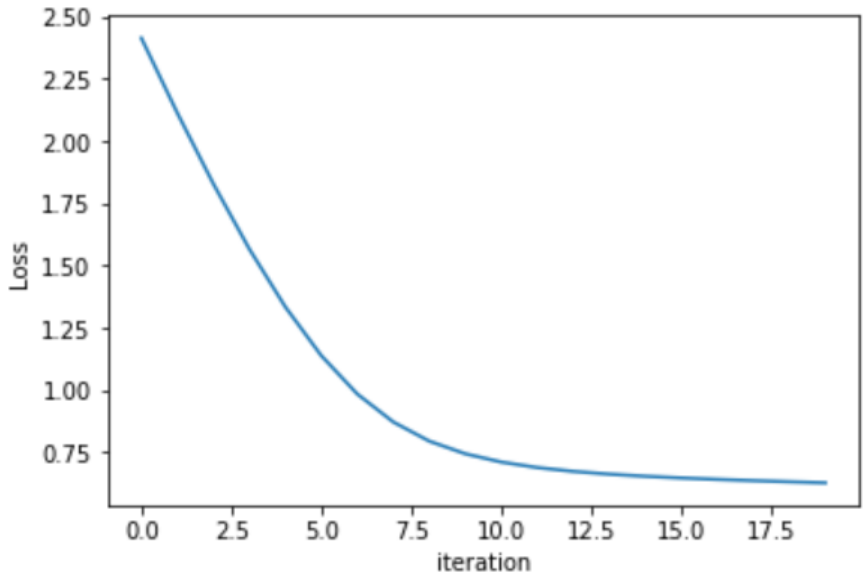
❖ Logistic regression

Feature	Label
Petal_Length	Category
1.4	0
1	0
1.5	0
3	1
3.8	1
4.1	1

Model

$$z = wx + b$$

$$\hat{y} = \frac{1}{1 + e^{-z}}$$




```python
import numpy as np
import tensorflow as tf
import tensorflow.keras as keras

batch_size = 6
epochs = 20

# Data Preparation
data = np.genfromtxt('iris_1D.csv', delimiter=',', skip_header=1)
X = data[:,0:1]
y = data[:,1]

# create model
model = tf.keras.Sequential(
    [tf.keras.layers.Dense(units=1, activation='sigmoid', input_shape=[1])])

# declare optimization method and loss function
opt = keras.optimizers.SGD(learning_rate=0.1)
model.compile(optimizer=opt, loss='binary_crossentropy')

# training
history = model.fit(X, y, batch_size, epochs)
```

```
Train on 6 samples
Epoch 1/20
6/6 [==============================] - 0s 20ms/sample - loss: 2.4125
Epoch 2/20
6/6 [==============================] - 0s 339us/sample - loss: 2.1118
Epoch 3/20
6/6 [==============================] - 0s 326us/sample - loss: 1.8278
Epoch 4/20
6/6 [==============================] - 0s 161us/sample - loss: 1.5661
Epoch 5/20
6/6 [==============================] - 0s 333us/sample - loss: 1.3337
Epoch 6/20
6/6 [==============================] - 0s 166us/sample - loss: 1.1381
Epoch 7/20
6/6 [==============================] - 0s 160us/sample - loss: 0.9842
```

Training

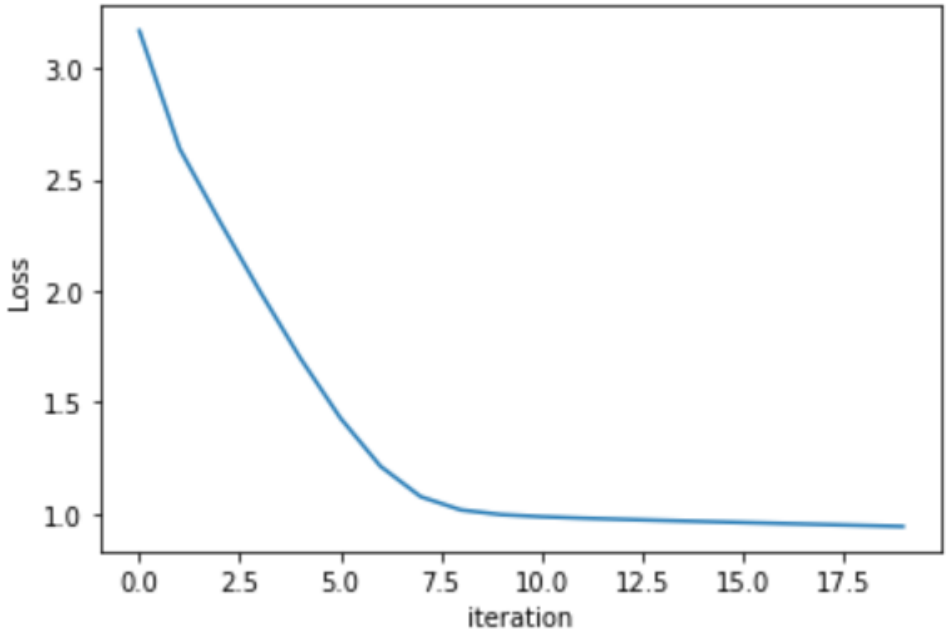
Petal_Length	Label
1.4	1
1.3	1
1.5	1
4.5	2
4.1	2
4.6	2
5.2	3
5.6	3
5.9	3

❖ Softmax regression

Model

$$z = \theta^T x$$

$$\hat{y} = \frac{e^z}{\sum_{i=1}^{k} e^{z_i}}$$




```python
import numpy as np
import tensorflow as tf
import tensorflow.keras as keras

batch_size = 9
epochs = 20

# Data Preparation
data = np.genfromtxt('iris_1D_3c.csv', delimiter=',', skip_header=1)
X = data[:,0:1]
y = data[:,1]

# create model
model = tf.keras.Sequential(
    [tf.keras.layers.Dense(units=3, activation='softmax', input_shape=[1])])

# declare optimization method and loss function
opt = keras.optimizers.SGD(learning_rate=0.1)
model.compile(optimizer=opt, loss='sparse_categorical_crossentropy')

# training
history = model.fit(X, y, batch_size, epochs)
```

```
Train on 9 samples
Epoch 1/20
9/9 [==============================] - 0s 19ms/sample - loss: 3.1676
Epoch 2/20
9/9 [==============================] - 0s 221us/sample - loss: 2.6389
Epoch 3/20
9/9 [==============================] - 0s 110us/sample - loss: 2.3121
Epoch 4/20
9/9 [==============================] - 0s 221us/sample - loss: 1.9978
Epoch 5/20
9/9 [==============================] - 0s 111us/sample - loss: 1.6993
Epoch 6/20
9/9 [==============================] - 0s 220us/sample - loss: 1.4296
Epoch 7/20
9/9 [==============================] - 0s 222us/sample - loss: 1.2127
Epoch 8/20
9/9 [==============================] - 0s 221us/sample - loss: 1.0761
Epoch 9/20
9/9 [==============================] - 0s 222us/sample - loss: 1.0165
Epoch 10/20
9/9 [==============================] - 0s 222us/sample - loss: 0.9956
```

Compile Function

Configures the model for training

```
compile(
        optimizer='rmsprop',
        loss=None,
        metrics=None,
        loss_weights=None,
        weighted_metrics=None,
        run_eagerly=None,
        steps_per_execution=None
)
```

```
# metrics

# compile
model.compile(optimizer='sgd',
              loss='sparse_categorical_crossentropy',
              metrics=[keras.metrics.SparseCategoricalAccuracy()])

# training
batch_size = 32
epochs = 5
history = model.fit(x, y, batch_size,
                    epochs=epochs, verbose=2)
```

```
Epoch 1/5
5/5 - 0s - loss: 0.4602 - sparse_categorical_accuracy: 0.9467 - 373ms/epoch - 75ms/step
Epoch 2/5
5/5 - 0s - loss: 0.4608 - sparse_categorical_accuracy: 0.8133 - 9ms/epoch - 2ms/step
```

Evaluate Function

```
evaluate(
        x=None,
        y=None,
        batch_size=None,
        verbose=1,
)
```

Returns the loss value & metrics values for the model in test mode

```
# compile
model.compile(optimizer='sgd',
              loss='sparse_categorical_crossentropy',
              metrics=[keras.metrics.SparseCategoricalAccuracy()])

# training
batch_size = 32
epochs = 500
history = model.fit(x_train, y_train, batch_size,
                    validation_data=(x_val, y_val),
                    epochs=epochs, verbose=0)

model.evaluate(x_val, y_val, batch_size, verbose=2)
```

```
1/1 - 0s - loss: 0.2789 - sparse_categorical_accuracy: 0.9333 - 29ms/epoch

[0.2789105176925659, 0.9333333373069763]
```

Fit Function

```python
# training
batch_size = 32
epochs = 500
history = model.fit(x, y, batch_size, epochs)
```

```
fit(
        x=None,
        y=None,
        batch_size=None,
        epochs=1,
        verbose='auto',
        callbacks=None,
        validation_split=0.0,
        validation_data=None,
        shuffle=True,
        class_weight=None,
        sample_weight=None,
        initial_epoch=0,
        steps_per_epoch=None,
        validation_steps=None,
        validation_batch_size=None,
        validation_freq=1,
        max_queue_size=10,
        workers=1,
        use_multiprocessing=False
)
```

Fit Function

```python
# compile
model.compile(optimizer='sgd',
              loss='sparse_categorical_crossentropy')

# training
batch_size = 32
epochs = 10
history = model.fit(X, y, batch_size,
                    epochs=epochs, initial_epoch=5)
```

```
Epoch 6/10
5/5 [==============================] - 0s 2ms/step - loss: 1.3554
Epoch 7/10
5/5 [==============================] - 0s 1ms/step - loss: 1.1875
Epoch 8/10
5/5 [==============================] - 0s 2ms/step - loss: 1.1136
Epoch 9/10
5/5 [==============================] - 0s 2ms/step - loss: 1.0656
Epoch 10/10
5/5 [==============================] - 0s 1ms/step - loss: 1.0340
```

```
fit(
    x=None,
    y=None,
    batch_size=None,
    epochs=1,
    verbose='auto',
    callbacks=None,
    validation_split=0.0,
    validation_data=None,
    shuffle=True,
    class_weight=None,
    sample_weight=None,
    initial_epoch=0,
    steps_per_epoch=None,
    validation_steps=None,
    validation_batch_size=None,
    validation_freq=1,
    max_queue_size=10,
    workers=1,
    use_multiprocessing=False
)
```

AI VIETNAM
All-in-One Course

Fit Function

verbose = 0	Silent
verbose = 1	Progress bar
verbose = 2	One line per epoch

```
fit(
        x=None,
        y=None,
        batch_size=None,
        epochs=1,
        verbose='auto',
)
```

```
# compile
model.compile(optimizer='sgd',
              loss='sparse_categorical_crossentropy')

# training
batch_size = 32
epochs = 6
history = model.fit(X, y, batch_size,
                    epochs=epochs, verbose=1)
```

```
Epoch 1/6
5/5 [==============================] - 0s 2ms/step - loss: 3.3299
Epoch 2/6
5/5 [==============================] - 0s 2ms/step - loss: 2.2940
Epoch 3/6
5/5 [==============================] - 0s 2ms/step - loss: 1.6176
Epoch 4/6
5/5 [==============================] - 0s 2ms/step - loss: 1.1914
Epoch 5/6
5/5 [==============================] - 0s 2ms/step - loss: 1.0308
Epoch 6/6
5/5 [==============================] - 0s 1ms/step - loss: 0.9764
```

```
# compile
model.compile(optimizer='sgd',
              loss='sparse_categorical_crossentropy')

# training
batch_size = 32
epochs = 6
history = model.fit(X, y, batch_size,
                    epochs=epochs, verbose=2)
```

```
Epoch 1/6
5/5 - 0s - loss: 0.8153 - 297ms/epoch - 59ms/step
Epoch 2/6
5/5 - 0s - loss: 0.7971 - 7ms/epoch - 1ms/step
Epoch 3/6
5/5 - 0s - loss: 0.7824 - 7ms/epoch - 1ms/step
Epoch 4/6
5/5 - 0s - loss: 0.7672 - 11ms/epoch - 2ms/step
Epoch 5/6
5/5 - 0s - loss: 0.7551 - 11ms/epoch - 2ms/step
Epoch 6/6
5/5 - 0s - loss: 0.7447 - 6ms/epoch - 1ms/step
```

AI VIETNAM
All-in-One Course

Fit Function

```python
# compile
model.compile(optimizer='sgd',
              loss='sparse_categorical_crossentropy')

# training
batch_size = 32
epochs = 6
history = model.fit(X, y, batch_size,
                    epochs=epochs, verbose=2,
                    validation_split=0.2)
```

```
Epoch 1/6
4/4 - 1s - loss: 0.8714 - val_loss: 0.7968 - 521ms/epoch - 130ms/step
Epoch 2/6
4/4 - 0s - loss: 0.8324 - val_loss: 0.8786 - 40ms/epoch - 10ms/step
Epoch 3/6
4/4 - 0s - loss: 0.8058 - val_loss: 0.9249 - 53ms/epoch - 13ms/step
Epoch 4/6
4/4 - 0s - loss: 0.7857 - val_loss: 0.9224 - 48ms/epoch - 12ms/step
Epoch 5/6
4/4 - 0s - loss: 0.7647 - val_loss: 0.9255 - 48ms/epoch - 12ms/step
Epoch 6/6
4/4 - 0s - loss: 0.7473 - val_loss: 0.9454 - 48ms/epoch - 12ms/step
```

```
fit(
    x=None,
    y=None,
    batch_size=None,
    epochs=1,
    verbose='auto',
    callbacks=None,
    validation_split=0.0,
    validation_data=None,
    shuffle=True,
    class_weight=None,
    sample_weight=None,
    initial_epoch=0,
    steps_per_epoch=None,
    validation_steps=None,
    validation_batch_size=None,
    validation_freq=1,
    max_queue_size=10,
    workers=1,
    use_multiprocessing=False
)
```

Fit Function

```python
import numpy as np

# Load data
data = np.genfromtxt('iris_full.csv',
                     delimiter=',',
                     skip_header=1)
x = data[:,0:4]
y = data[:,4:]
N = x.shape[0]

# shuffle
inds = np.arange(N)
np.random.shuffle(inds)

x = x[inds]
y = y[inds]

# prepare train and val data
N_train = 120
x_train = x[:N_train]
y_train = y[:N_train]

x_val = x[N_train:]
y_val = y[N_train:]
```

```
fit(
    x=None,
    y=None,
    batch_size=None,
    epochs=1,
    verbose='auto',
    callbacks=None,
    validation_split=0.0,
    validation_data=None,
    shuffle=True,
    class_weight=None,
    sample_weight=None,
    initial_epoch=0,
    steps_per_epoch=None,
    validation_steps=None,
    validation_batch_size=None,
    validation_freq=1,
    max_queue_size=10,
    workers=1,
    use_multiprocessing=False
)
```

AI VIETNAM
All-in-One Course

Fit Function

```python
# compile
model.compile(optimizer='sgd',
              loss='sparse_categorical_crossentropy')

# training
batch_size = 32
epochs = 6
history = model.fit(x_train, y_train, batch_size,
                    validation_data=(x_val, y_val),
                    epochs=epochs, verbose=2)
```

```
Epoch 1/6
4/4 - 1s - loss: 1.3266 - val_loss: 1.0424 - 512ms/epoch - 128ms/step
Epoch 2/6
4/4 - 0s - loss: 0.9921 - val_loss: 0.7892 - 36ms/epoch - 9ms/step
Epoch 3/6
4/4 - 0s - loss: 0.7719 - val_loss: 0.6777 - 41ms/epoch - 10ms/step
Epoch 4/6
4/4 - 0s - loss: 0.6949 - val_loss: 0.6468 - 41ms/epoch - 10ms/step
Epoch 5/6
4/4 - 0s - loss: 0.6648 - val_loss: 0.6351 - 42ms/epoch - 10ms/step
Epoch 6/6
4/4 - 0s - loss: 0.6501 - val_loss: 0.6252 - 40ms/epoch - 10ms/step
```

```
fit(
    x=None,
    y=None,
    batch_size=None,
    epochs=1,
    verbose='auto',
    callbacks=None,
    validation_split=0.0,
    validation_data=None,
    shuffle=True,
    class_weight=None,
    sample_weight=None,
    initial_epoch=0,
    steps_per_epoch=None,
    validation_steps=None,
    validation_batch_size=None,
    validation_freq=1,
    max_queue_size=10,
    workers=1,
    use_multiprocessing=False
)
```

Fit Function

```python
# callbacks

# compile
model.compile(optimizer='sgd',
              loss='sparse_categorical_crossentropy')

# training
callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath="models/model_{epoch}",
        save_best_only=True,
        monitor="val_loss",
        verbose=1
    )
]

batch_size = 32
epochs = 5
history = model.fit(x_train, y_train, batch_size,
                    validation_data=(x_val, y_val),
                    epochs=epochs, verbose=2,
                    callbacks=callbacks)
```

```
fit(
        x=None,
        y=None,
        batch_size=None,
        epochs=1,
        verbose='auto',
        callbacks=None,
        validation_split=0.0,
        validation_data=None,
        shuffle=True,
        class_weight=None,
        sample_weight=None,
        initial_epoch=0,
        steps_per_epoch=None,
        validation_steps=None,
        validation_batch_size=None,
        validation_freq=1,
        max_queue_size=10,
        workers=1,
        use_multiprocessing=False
)
```

Predict Function

```
predict(
        x=None,
        y=None,
        batch_size=None,
        verbose=1,
)
```

Generates output predictions
for the input samples

```
# compile
model.compile(optimizer='sgd',
              loss='sparse_categorical_crossentropy',
              metrics=[keras.metrics.SparseCategoricalAccuracy()])

# training
batch_size = 32
epochs = 500
history = model.fit(x_train, y_train, batch_size,
                    validation_data=(x_val, y_val),
                    epochs=epochs, verbose=0)

model.evaluate(x_val, y_val, batch_size, verbose=2)
```

```
1/1 - 0s - loss: 0.2789 - sparse_categorical_accuracy: 0.9333 - 29ms/epoch

[0.2789105176925659, 0.9333333373069763]
```

```
output = model.predict(x_val)
print(output.shape)
```

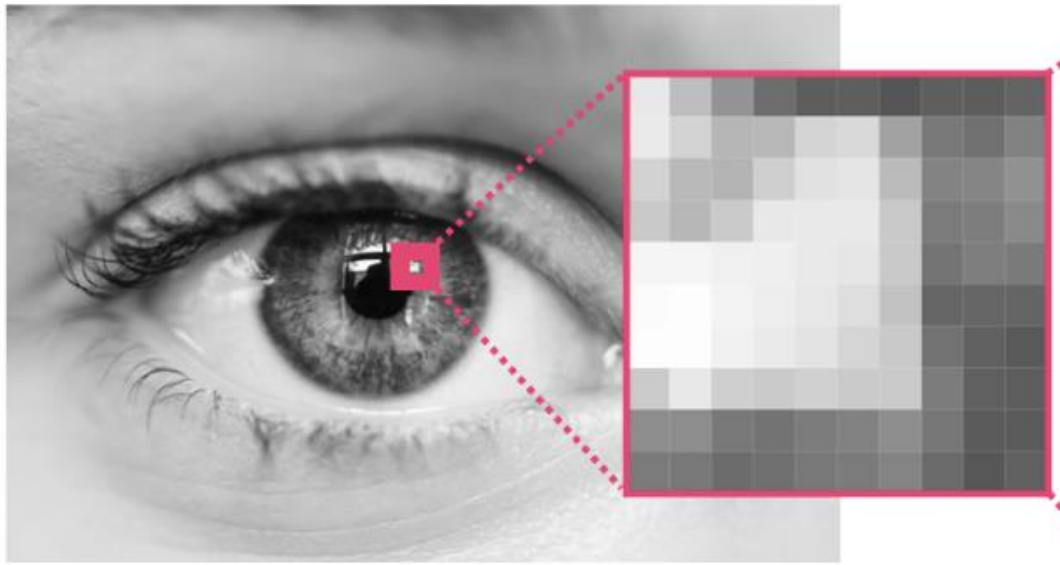
```
(30, 3)
```

Outline

- Introduction to Tensorflow
- Introduction to Tensforflow.Keras
- Model Construction
- Model Training and Inference
- Applying Softmax for Image Data

Image Classification: Image Data

❖ Grayscale images



230	194	147	108	90	98	84	96	91	101
237	206	188	195	207	213	163	123	116	128
210	183	180	205	224	234	188	122	134	147
198	189	201	227	229	232	200	125	127	135
249	241	237	244	232	226	202	116	125	126
251	254	241	239	230	217	196	102	103	99
243	255	240	231	227	214	203	116	95	91
204	231	208	200	207	201	200	121	95	95
144	140	120	115	125	127	143	118	92	91
121	121	108	109	122	121	134	106	86	97

(Height, Width)

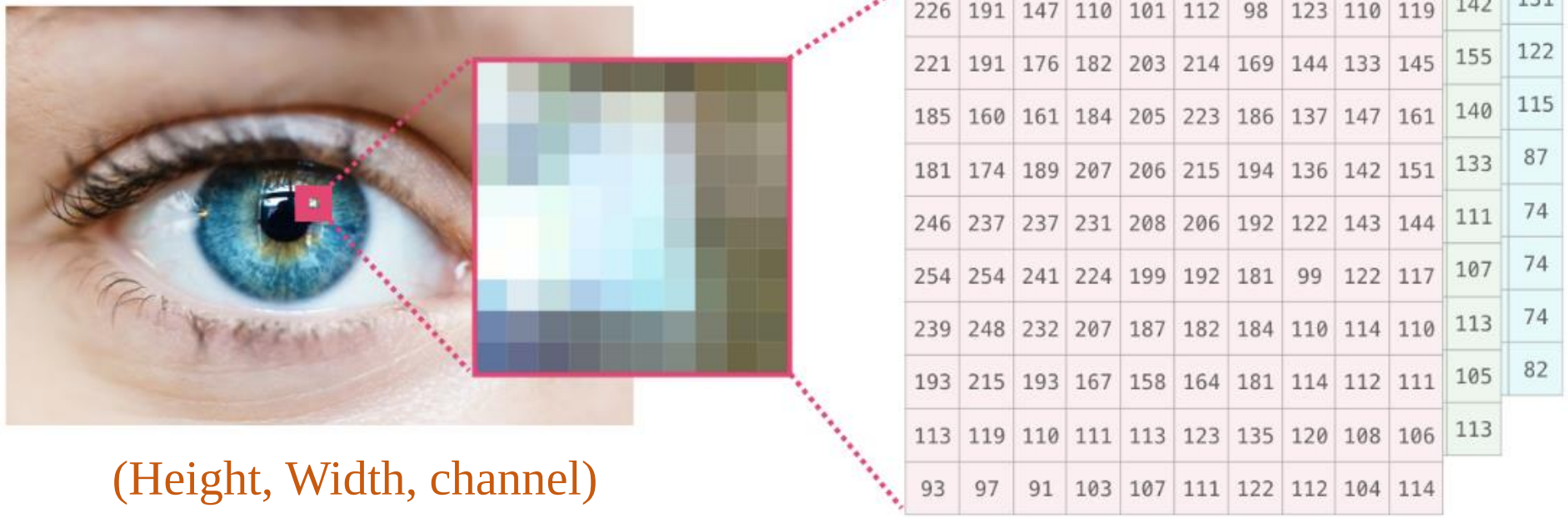
Pixel p = scalar

$0 \leq p \leq 255$

Resolution: #pixels
Resolution = HeightxWidth

Image Classification: Image Data

❖ Color images



(Height, Width, channel)

RGB color image

Pixel $p = \begin{bmatrix} r \\ g \\ b \end{bmatrix}$

$0 \leq r,g,b \leq 255$

Resolution: #pixels
Resolution = HeightxWidth

Important Packages

❖ Some functions

To download a file

```
import urllib.request as req
req.urlretrieve(url, name)
```

To open an image

```
from PIL import Image
img = Image.open(name)
```

To show an image

```
import matplotlib.pyplot as plt
plt.imshow(img)
```

```
import urllib.request as req
from PIL import Image
import matplotlib.pyplot as plt

# download an image
req.urlretrieve('https://www.dropbox.com/s/zwy8ddkdm3thatr/nature.jpg?dl=1',
                'image.jpg')

# show the image
img = Image.open('image.jpg')
plt.imshow(img)
```

```
<matplotlib.image.AxesImage at 0x7f5088018b90>
```

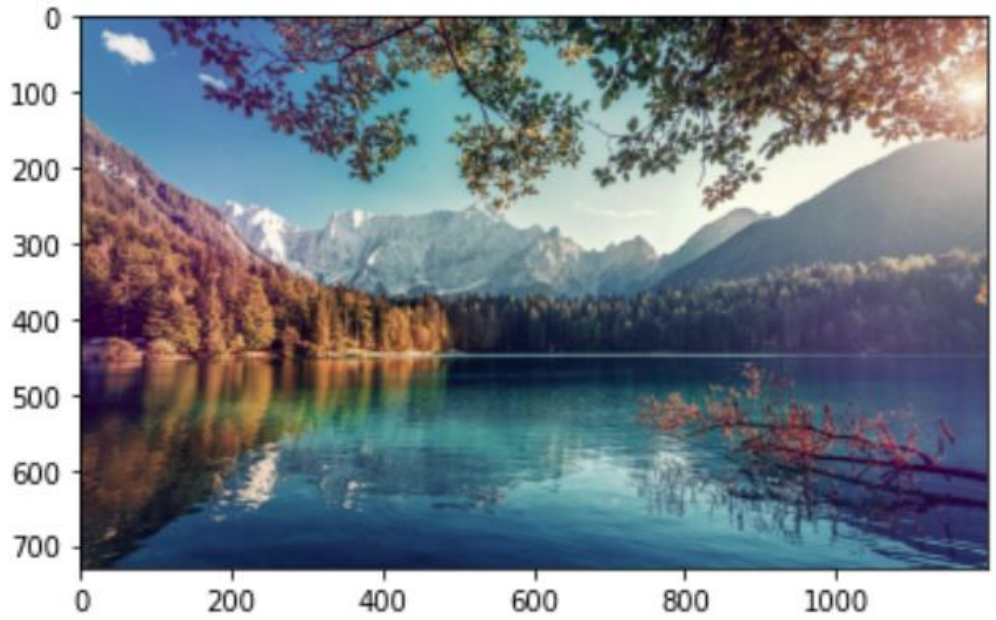


Image Data

MNIST dataset

- Grayscale images
- Resolution=28x28
- Training set: 60000 samples
- Testing set: 10000 samples

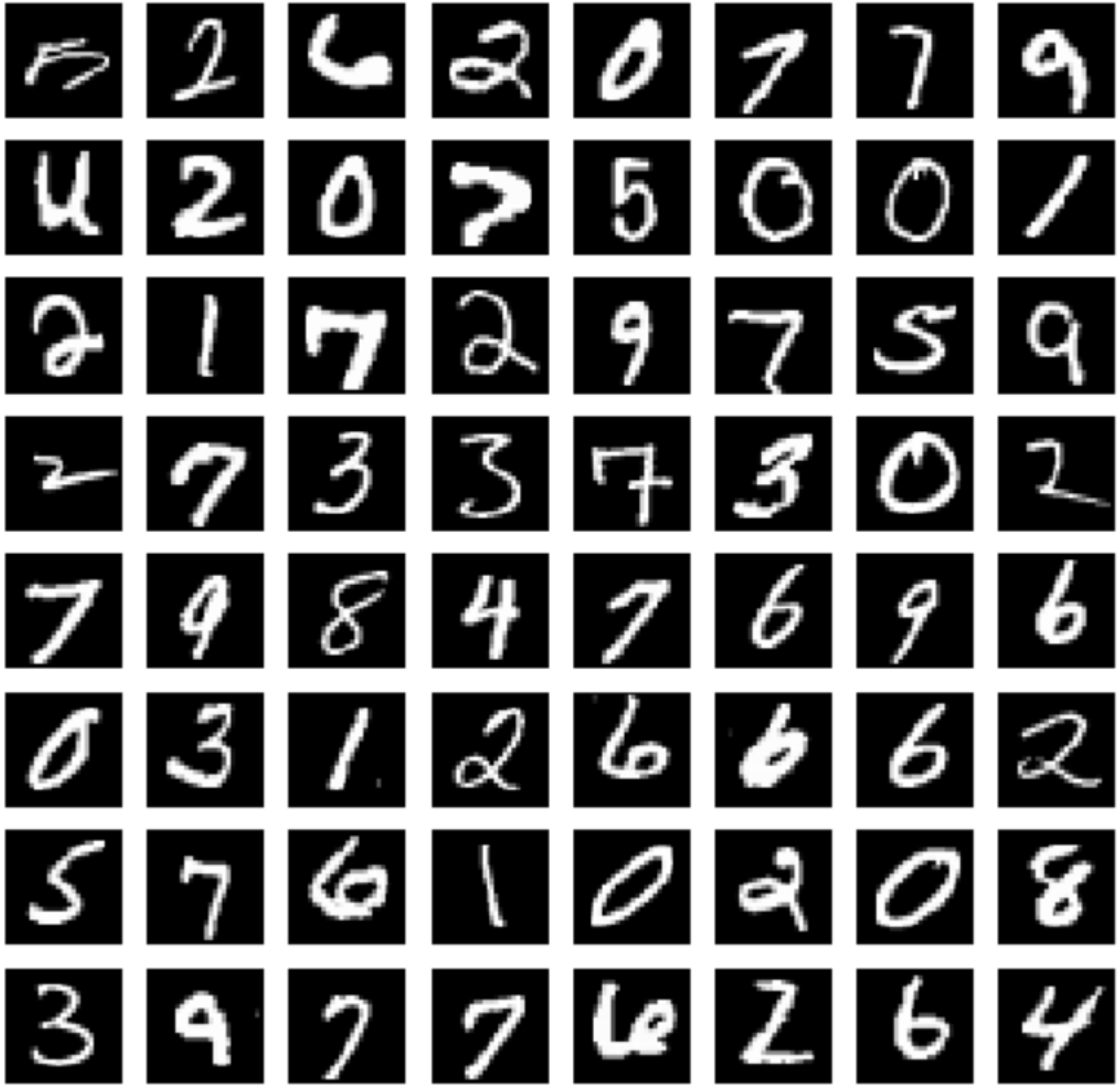


Image Data

MNIST dataset

- Grayscale images
- Resolution=28x28
- Training set: 60000 samples
- Testing set: 10000 samples

TRAINING SET LABEL FILE (train-labels-idx1-ubyte):

[offset]	[type]	[value]	[description]
0000	32 bit integer	0x00000801(2049)	magic number (MSB first)
0004	32 bit integer	60000	number of items
0008	unsigned byte	??	label
0009	unsigned byte	??	label
.....			
xxxx	unsigned byte	??	label

The labels values are 0 to 9.

TRAINING SET IMAGE FILE (train-images-idx3-ubyte):

[offset]	[type]	[value]	[description]
0000	32 bit integer	0x00000803(2051)	magic number
0004	32 bit integer	60000	number of images
0008	32 bit integer	28	number of rows
0012	32 bit integer	28	number of columns
0016	unsigned byte	??	pixel
0017	unsigned byte	??	pixel
.....			
xxxx	unsigned byte	??	pixel

http://yann.lecun.com/exdb/mnist/

Image Data

```python
def load_mnist():
    mnist = {}
    for name in filename[:2]:
        with gzip.open(folder+name[1], 'rb') as f:
            mnist[name[0]] = np.frombuffer(f.read(), np.uint8, offset=16).reshape(-1,28*28)

    for name in filename[-2:]:
        with gzip.open(folder+name[1], 'rb') as f:
            mnist[name[0]] = np.frombuffer(f.read(), np.uint8, offset=8)

    return mnist["training_images"], mnist["training_labels"], mnist["test_images"], mnist["test_labels"]

X_train, y_train, X_test, y_test = load_mnist()

#kiểm tra dự liệu
print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
```

```
(60000, 784)
(60000,)
(10000, 784)
(10000,)
```

MNIST dataset

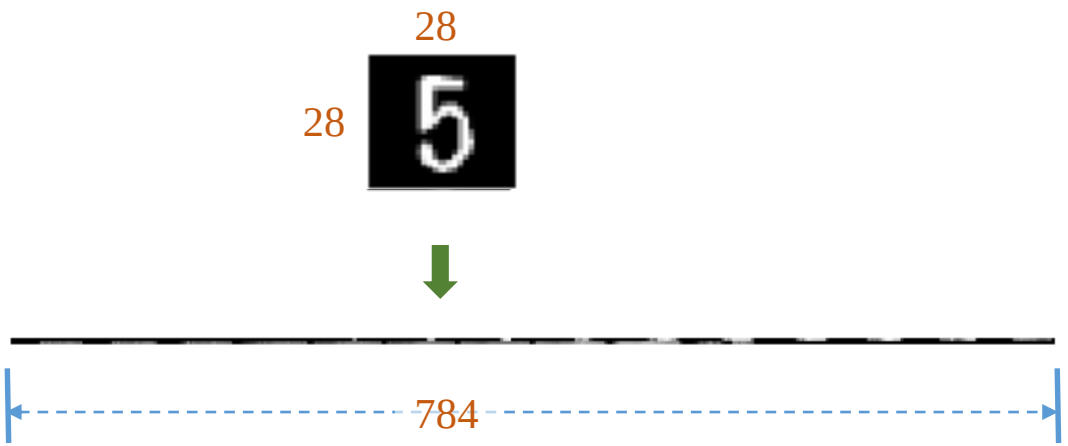


Image Data

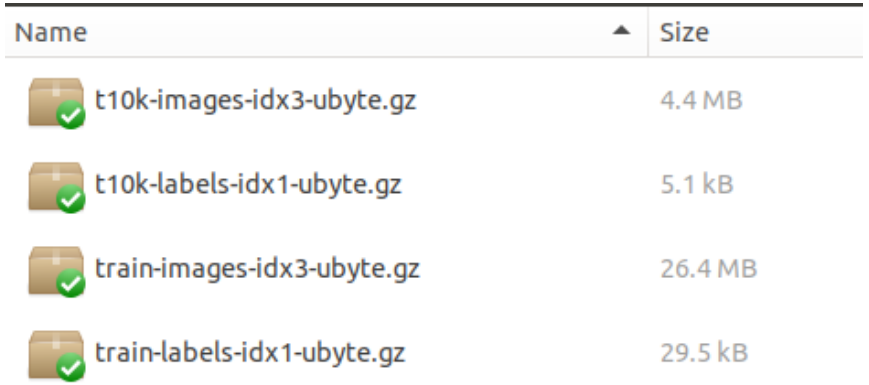
Fashion-MNIST dataset

- Grayscale images
- Resolution=28x28
- Training set: 60000 samples
- Testing set: 10000 samples

Image Classification

❖ Fashion-MNIST data

Download data

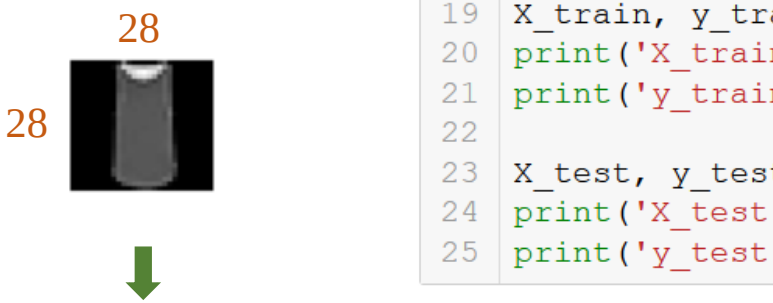



```python
import numpy as np
from urllib import request
import gzip
import pickle

filename = [["training_images","train-images-idx3-ubyte.gz"],
            ["test_images","train-labels-idx1-ubyte.gz"],
            ["training_labels","t10k-images-idx3-ubyte.gz"],
            ["test_labels","t10k-labels-idx1-ubyte.gz"]]

# function to download data
def download_fashion_mnist(folder):
    base_url = "http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/"
    for name in filename:
        print("Downloading " + name[1] + "...")

        # luu vào folder data_fashion_mnist
        request.urlretrieve(base_url + name[1], folder + name[1])
    print("Download complete.")

# download dataset và save to folder 'data_fashion_mnist/'
folder = 'data_fashion_mnist/'
download_fashion_mnist(folder)
```

Image Classification

Fashion-MNIST data

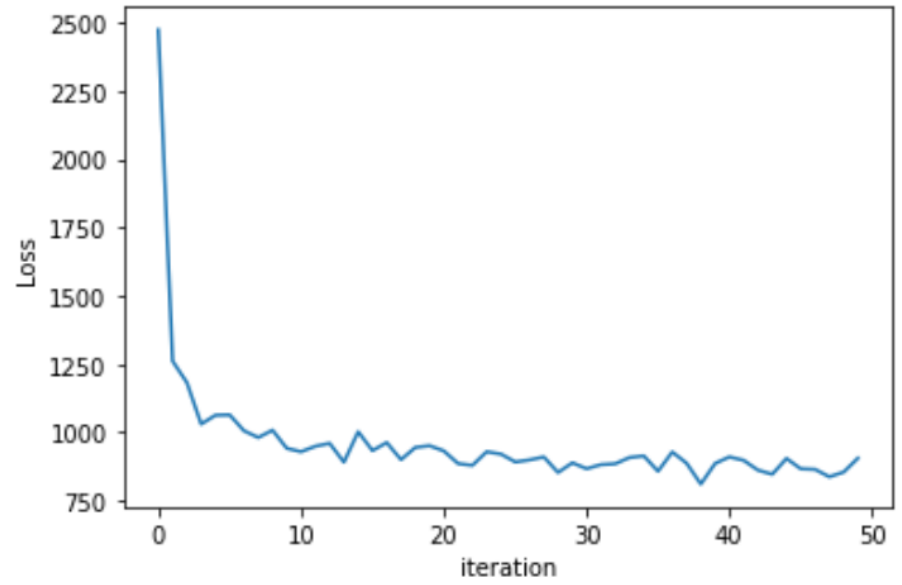
```
X_train: (60000, 784)
y_train: (60000,)
X_test: (10000, 784)
y_test: (10000,)
```

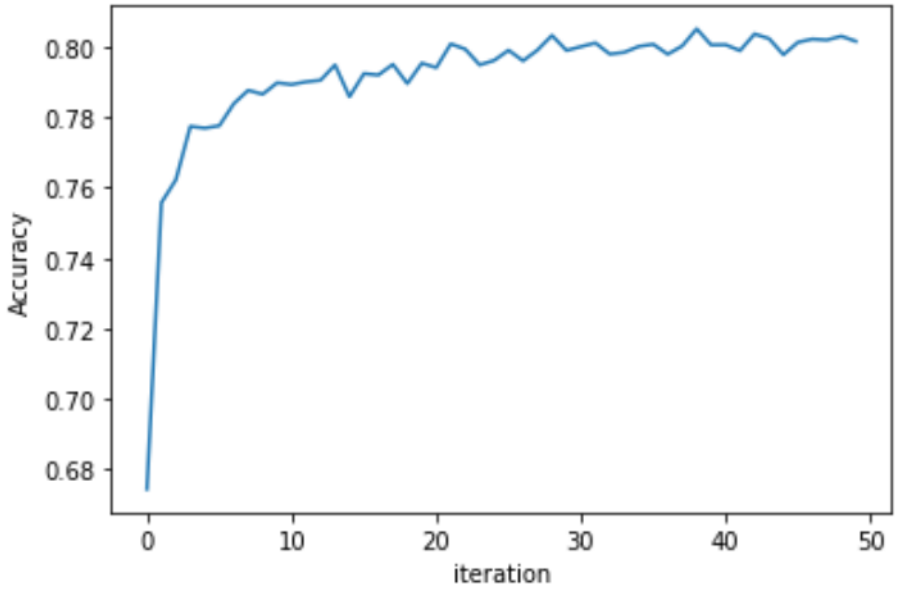


Read data

```python
import os
import gzip
import numpy as np

def load_fashion_mnist(path, kind='train'):
    """Load fashion_MNIST data from `path`"""
    labels_path = os.path.join(path, '%s-labels-idx1-ubyte.gz' % kind)
    images_path = os.path.join(path, '%s-images-idx3-ubyte.gz' % kind)

    with gzip.open(labels_path, 'rb') as lbpath:
        labels = np.frombuffer(lbpath.read(), dtype=np.uint8, offset=8)
    with gzip.open(images_path, 'rb') as imgpath:
        images = np.frombuffer(imgpath.read(),
                               dtype=np.uint8, offset=16).reshape(len(labels), 784)

    return images, labels


X_train, y_train = load_fashion_mnist('C:/Data/data_fashion_mnist/')
print('X_train:', X_train.shape)
print('y_train:', y_train.shape)

X_test, y_test = load_fashion_mnist('C:/Data/data_fashion_mnist/', kind='t10k')
print('X_test:', X_test.shape)
print('y_test:', y_test.shape)
```

Demo






```python
import tensorflow as tf
import tensorflow.keras as keras

# create model
model = keras.Sequential()
model.add(keras.Input(shape=(784,)))
model.add(keras.layers.Dense(10, activation='softmax'))
model.summary()

# optimizer and loss
model.compile(optimizer='sgd',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# training
history = model.fit(X_train, y_train, 256,
                    epochs=50, verbose=2)

# testing
test_loss, test_acc = model.evaluate(X_test, y_test)
print('Test accuracy:', test_acc)
```

Demo

```python
import tensorflow as tf
import tensorflow.keras as keras

# predict
def predict(x, W, b):
    return tf.math.softmax(tf.matmul(x, W) + b)

# weights
W = tf.Variable(tf.random.normal((784, 10),
                                 mean=0,
                                 stddev=0.01,
                                 dtype=tf.float64))
b = tf.Variable(tf.zeros((10,),
                         dtype=tf.float64))

# loss function
loss_fn = keras.losses.SparseCategoricalCrossentropy()
```

GradientTape

```python
# training
lr = 0.01
num_epochs = 50
losses = [] # for debug

for epoch in range(num_epochs):
    with tf.GradientTape() as t:
        # output
        output = predict(X_train, W, b)

        # loss
        loss = loss_fn(y_train, output)
        losses.append(loss.numpy())

    # gradient
    dW, db = t.gradient(loss, [W, b])

    # update
    W.assign_sub(lr*dW)
    b.assign_sub(lr*db)
```

Model Saving and Loading

Model Saving

```python
import numpy as np
import tensorflow as tf
import tensorflow.keras as keras

batch_size = 4
epochs = 10

# Data Preparation
data = np.genfromtxt('data.csv', delimiter=',')
X = data[:,0:1]
y = data[:,1:]

# create model
model = tf.keras.Sequential(
    [tf.keras.layers.Dense(units=1, input_shape=[1])])

# declare optimization method and loss function
opt = keras.optimizers.SGD(learning_rate=0.01)
model.compile(optimizer=opt, loss='mse')

# training
history = model.fit(X, y, batch_size, epochs)

# save entire model
model.save('my_model/model.h5')
```

Model Loading

```python
import tensorflow as tf
import tensorflow.keras as keras

# load model
model = tf.keras.models.load_model('my_model/model.h5')

# testing
X_testing = [[5.0]]
y_hat = model.predict(X_testing)
print(y_hat)
```

```
[[6.5115185]]
```

Tensorflow

❖ Demo

```
Python 3.7.3 (default, Apr 24 2019, 15:29:51) [MSC v.1915 64 bit (AMD64)] ::
Type "help", "copyright", "credits" or "license" for more information.
>>>
>>>
>>>
>>>
>>>
>>>
>>>
>>> for epoch in range(n_epochs):
...     sum_of_losses = 0
...     gradients = np.zeros((2,1))
...
...     for index in range(4):
...         xi = X_b[index:index+1]
...         yi = y[index:index+1]
```

Reference

Tensor

https://www.tensorflow.org/guide/tensor

TensorFlow 2 quickstart for beginners

https://www.tensorflow.org/tutorials/quickstart/beginner

Save and load models

https://www.tensorflow.org/tutorials/keras/save_and_load

Gradient tape

https://www.tensorflow.org/guide/autodiff