COMP7035

Python for Data Analytics and Artificial Intelligence

Keras

Renjie Wan

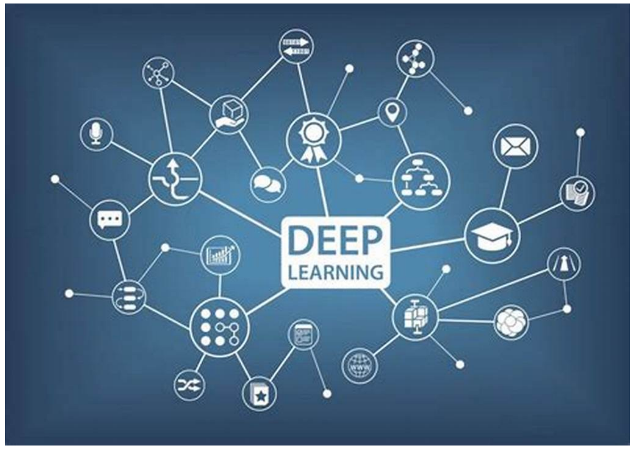
25/11/2024

香 港 浸 會 大 學
HONG KONG BAPTIST UNIVERSITY

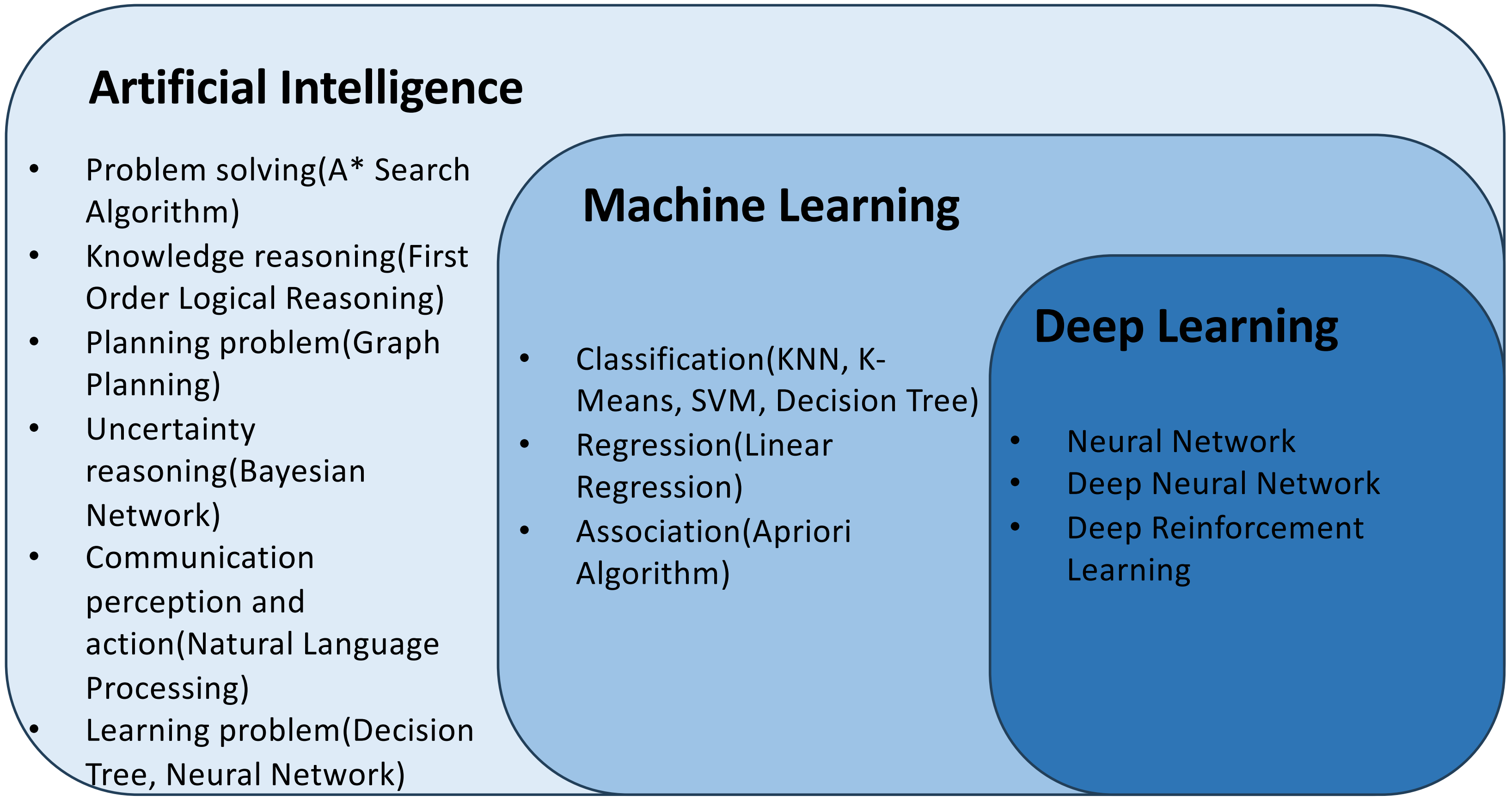
DEPARTMENT OF
COMPUTER SCIENCE
計算機科學系

Briefings on Deep Learning

- Deep Learning is a new research direction in the field of Machine Learning. It has been introduced into machine learning to bring it closer to its original goal—Artificial Intelligence.
- Its ultimate goal is to enable machines to have the analytical and learning capabilities similar to humans, allowing them to recognize and interpret data like text, images, and sounds.



Briefings on Deep Learning



Artificial Intelligence

- Problem solving(A* Search Algorithm)
- Knowledge reasoning(First Order Logical Reasoning)
- Planning problem(Graph Planning)
- Uncertainty reasoning(Bayesian Network)
- Communication perception and action(Natural Language Processing)
- Learning problem(Decision Tree, Neural Network)

Machine Learning

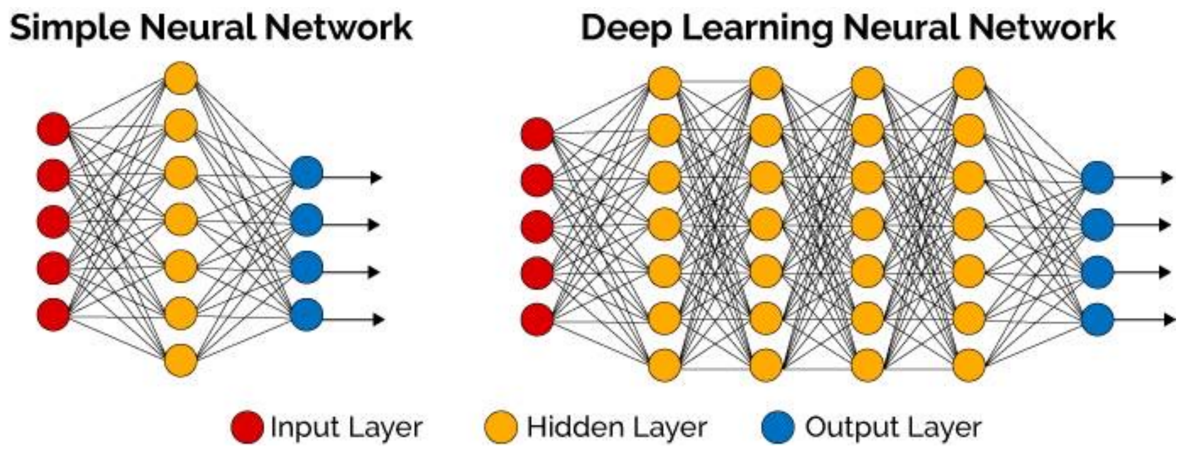
- Classification(KNN, K-Means, SVM, Decision Tree)
- Regression(Linear Regression)
- Association(Apriori Algorithm)

Deep Learning

- Neural Network
- Deep Neural Network
- Deep Reinforcement Learning

Briefings on Deep Learning

- In Deep Learning, the term **"Deep"** refers to the use of multiple layers in the neural network architecture.
- Traditional machine learning models often rely on shallow networks with only one or a few layers, which limits their capacity to learn complex patterns.
- In contrast, deep learning models consist of many layers—sometimes even dozens or hundreds—that are stacked on top of one another.



Briefings on Deep Learning

- Deep learning has achieved significant results in various fields, including search technology, data mining, machine learning, machine translation, natural language processing, multimedia learning, speech, recommendation and personalization technologies, among others.

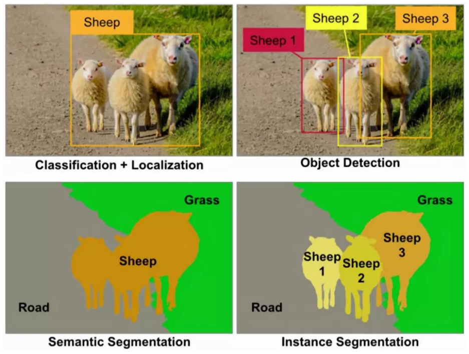
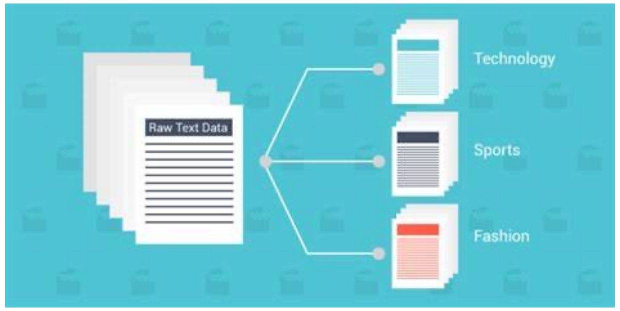
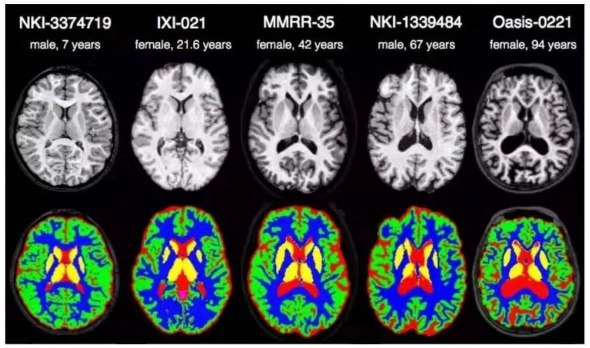


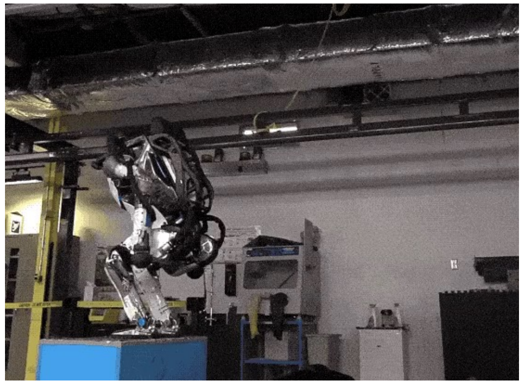
Image Processing



Text Classification



Medical Health Diagnosis



Robot Control

What is Keras?

- Keras is a powerful and easy-to-use free open source Python library for developing and evaluating deep learning (DL) models.
- It is part of the TensorFlow library and allows you to define and train neural network models easily.
- While many DL frameworks are available, the pipelines are very similar. Here we take Keras as an example.

TensorFlow Keras PyTorch

Top 3 Deep Learning Frameworks

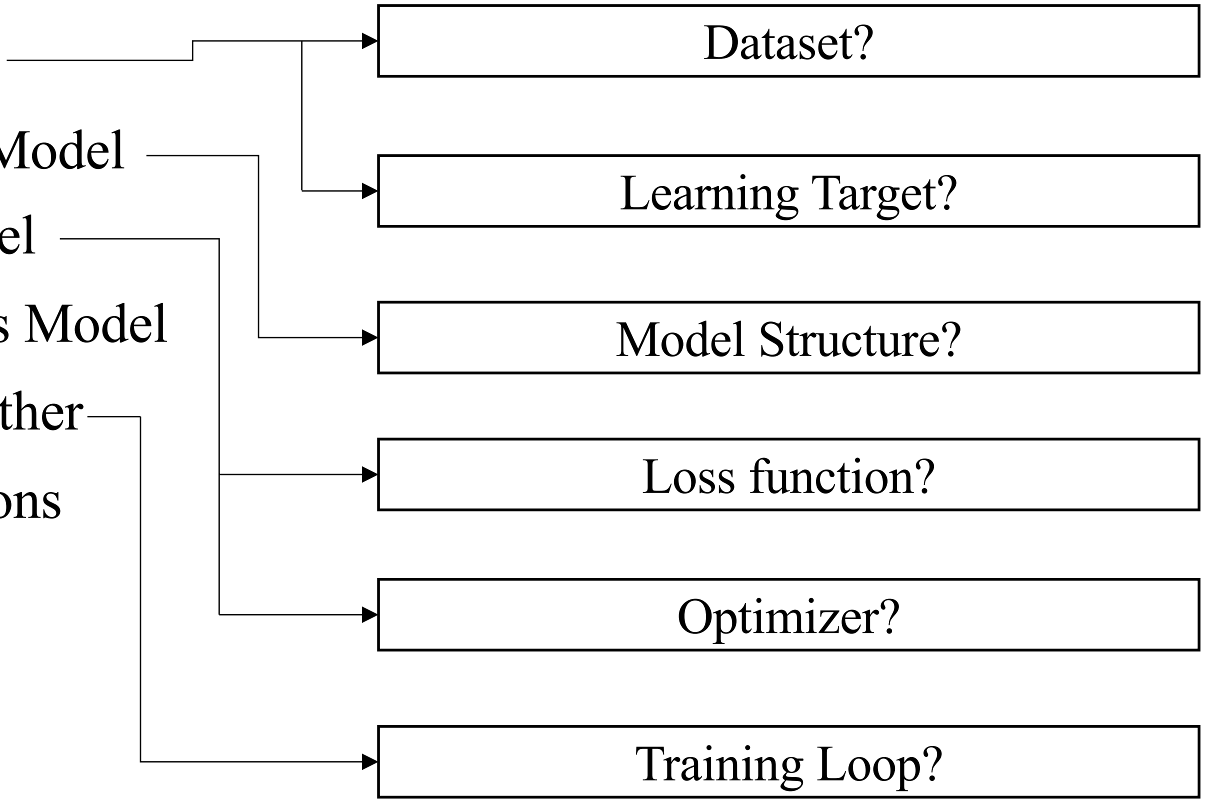
Install and Import Keras

- In Colab, run

```
! pip install keras
import keras
```

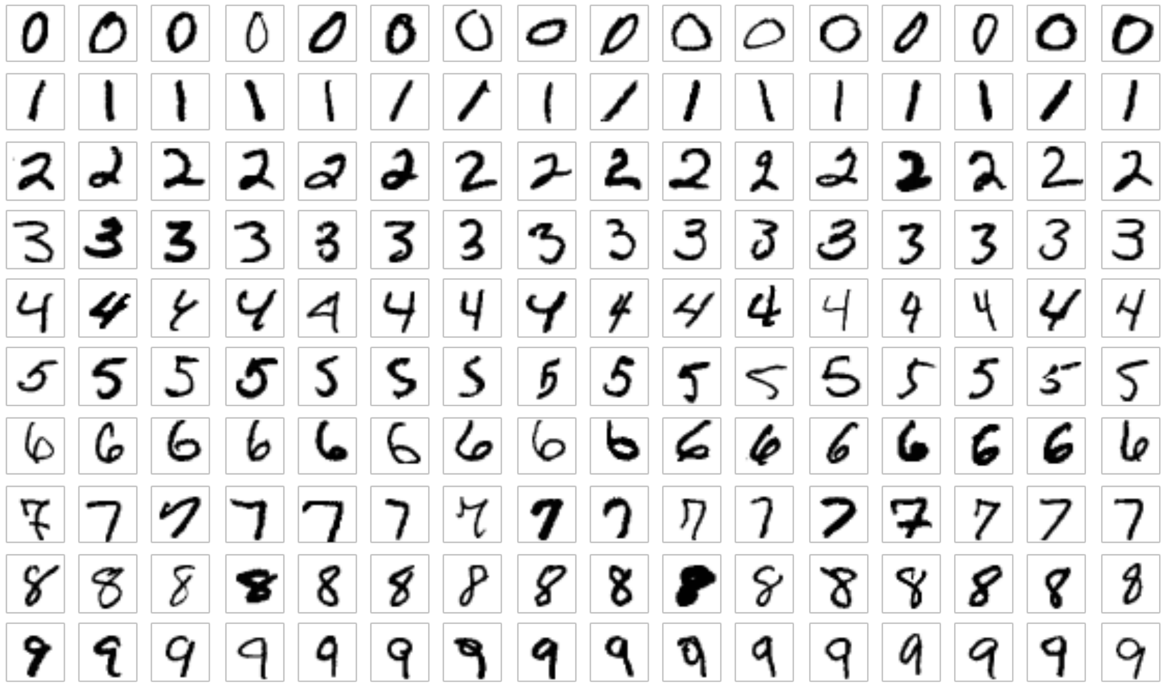
Key Components of Keras Pipeline

1. Load Data
2. Define Keras Model
3. Fit Keras Model
4. Evaluate Keras Model
5. Tie It All Together
6. Make Predictions



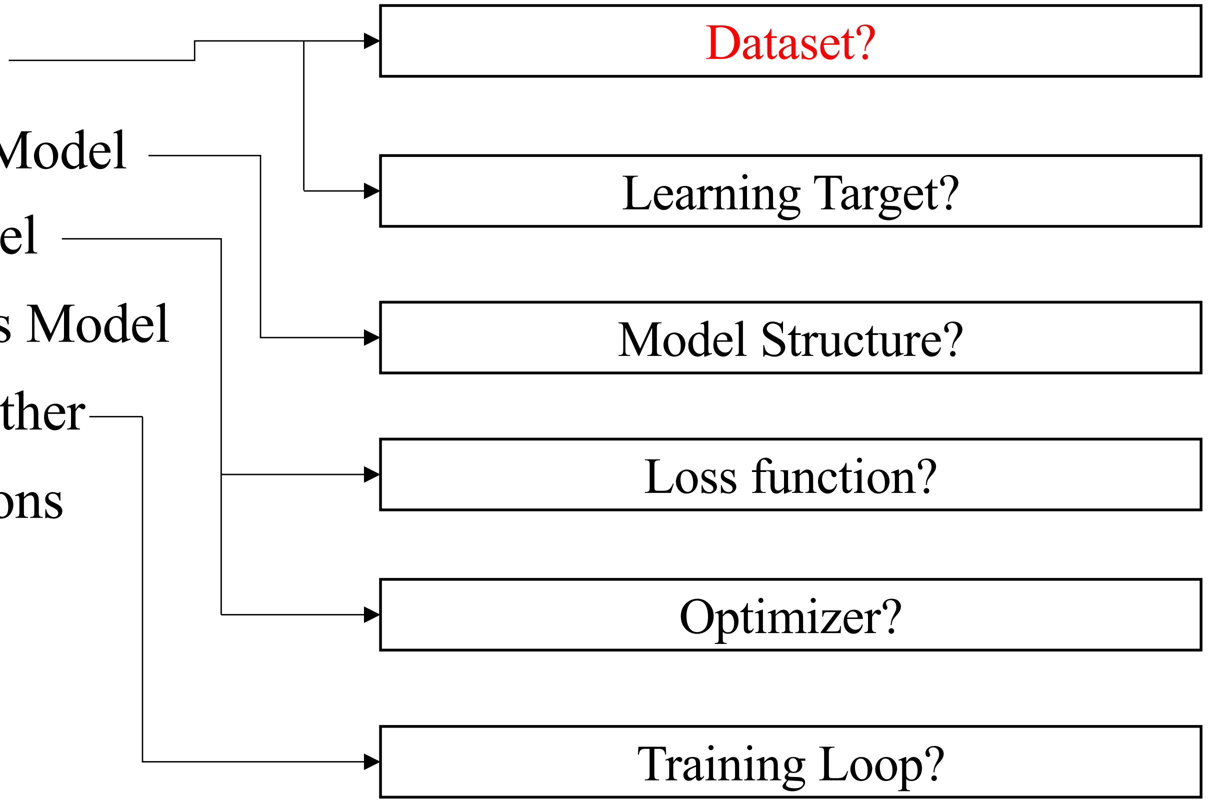
Key Components of Keras Pipeline

- We will learn each step based on a famous task:
  - MNIST handwritten digits recognition



Key Components of Keras Pipeline

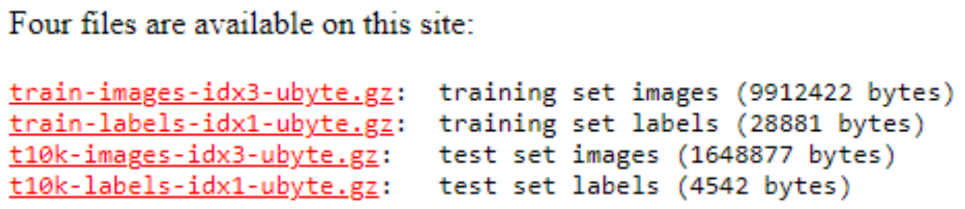
1. Load Data
2. Define Keras Model
3. Fit Keras Model
4. Evaluate Keras Model
5. Tie It All Together
6. Make Predictions



Loading Datasets

- Let us first download the data from the web:

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



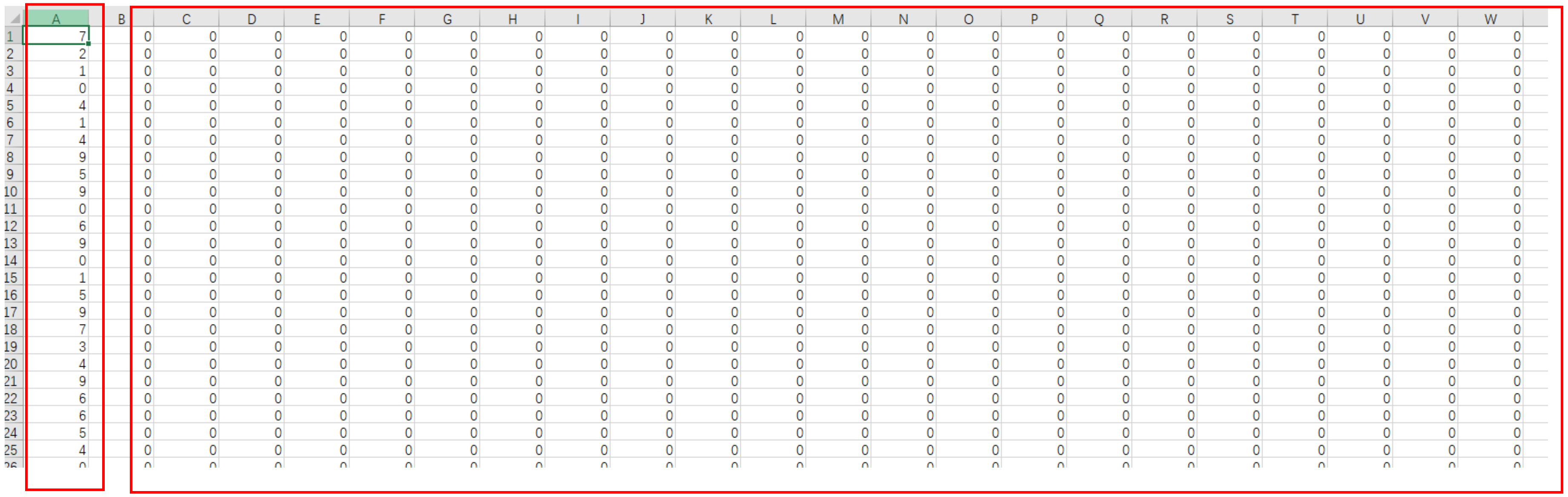
Four files are available on this site:

```
train-images-idx3-ubyte.gz:  training set images (9912422 bytes)
train-labels-idx1-ubyte.gz:  training set labels (28881 bytes)
t10k-images-idx3-ubyte.gz:   test set images (1648877 bytes)
t10k-labels-idx1-ubyte.gz:   test set labels (4542 bytes)
```

These files contains the pixel values and labels for each digit image.
The pixel values range from 0 to 255.

Loading Datasets

- Using the codes in convert.py to convert the raw dataset to the csv files, we can check the details of the dataset:



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
1	7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
16	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
17	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
18	7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
19	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
20	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
21	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
22	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
23	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
24	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
25	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Pixel values (features)

Try converting the dataset by yourself and check the csv files.

Loading Datasets

- Although there are some wrapped API to load the MNIST dataset, we will introduce **the general ways of loading a dataset**.

1. Loading the entire dataset when there is enough memory
2. Define a customized data generator to load the data sample one by one.

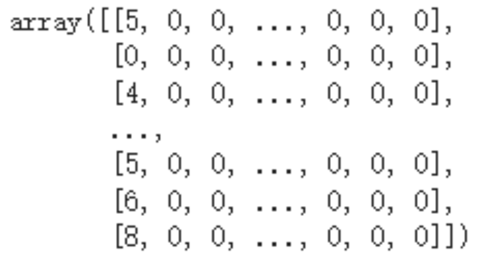
Loading Datasets

Loading the entire dataset

Because the MNIST dataset is small, we can load all the data into the memory

- We have created two csv files: mnist_train.csv and mnist_test.csv.
- Now we use Pandas to load the csv files and extract the values.

```
import pandas as pd
df_orig_train = pd.read_csv('mnist_train.csv',header=None)
df_orig_test = pd.read_csv('mnist_test.csv',header=None)
df_train_values = df_orig_train.values
df_test_values = df_orig_test.values
```



```
array([[5, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [4, 0, 0, ..., 0, 0, 0],
       ...,
       [5, 0, 0, ..., 0, 0, 0],
       [6, 0, 0, ..., 0, 0, 0],
       [8, 0, 0, ..., 0, 0, 0]])
```

Loading Datasets

Loading the entire dataset

Then we extract the features and labels for the training and test set. Since the pixel values are in the range of [0,255], we normalize them to be within the range of [0,1]

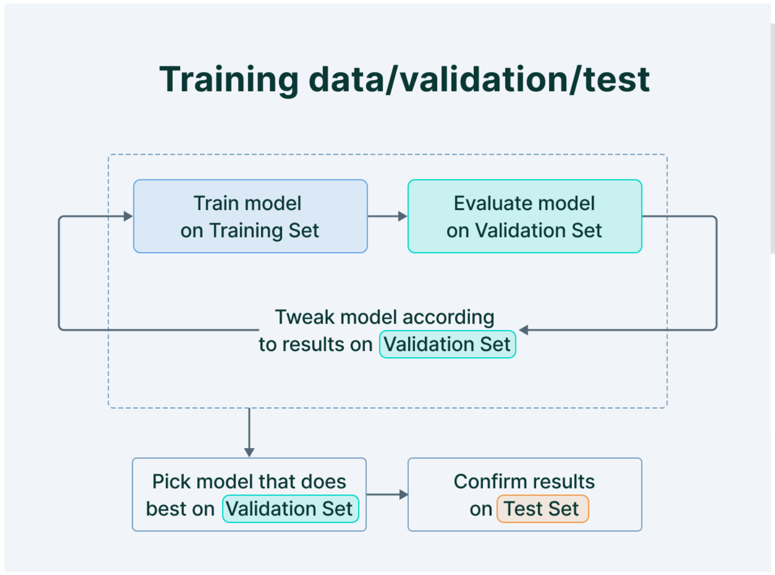
```
train_feat_ori,train_label_ori = df_train_values[:,1:]/255.0,df_train_values[:,0]
test_feat,test_label = df_test_values[:,1:]/255.0,df_test_values[:,0]
# show the shape of each variable
for i_variable in [train_feat_ori,train_label_ori,test_feat,test_label]:
  print(i_variable.shape)
```

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

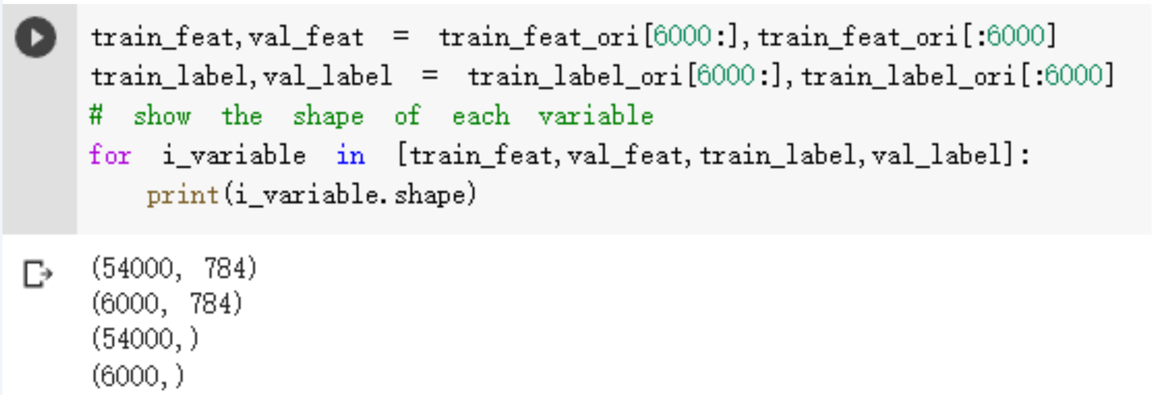
Loading Datasets

Loading the entire dataset

We also want to create a "validation" dataset, in order to verify whether the model has a good generalization ability during training.



Training data/validation/test
Train model on Training Set
Evaluate model on Validation Set
Tweak model according to results on Validation Set
Pick model that does best on Validation Set
Confirm results on Test Set



```
train_feat,val_feat  =  train_feat_ori[6000:],train_feat_ori[:6000]
train_label,val_label  =  train_label_ori[6000:],train_label_ori[:6000]
# show the shape of each variable
for  i_variable  in  [train_feat,val_feat,train_label,val_label]:
    print(i_variable.shape)
```

```
(54000, 784)
(6000, 784)
(54000,)
(6000,)
```

Here we take 6000 samples out of the original training set as the validation set.

Loading Datasets

Defining a data generator

In many cases the dataset is huge such that it is impossible to load the entire dataset into the memory before training.

- Speech Recognition: 20,000 hours of data

We can define a customized "data generator" to specify the rules of creating one sample.

Loading Datasets

Defining a data generator

The data generator is implemented as a class, which has three key methods:

__init__, __len__, and __getitem__.

```
class DataGenerator(keras.utils.Sequence):
    'Generates data for Keras'
    def __init__(self, para1, para2):
        # initilizes some variables according to the input paramet
ers.
        # Any number of parameters are supported
        # some codes
    def __len__(self):
        # return the total number of samples in the dataset
        # some codes
    def __getitem__(self,index):
        # create one sample according to the index
        # some codes
```

Loading Datasets

Defining a data generator

__init__

Instead of loading all the data, we use the directory of the csv file as the input, and let the generator to read only one line of the file when generating each sample (supported by linecache library).

```python
import random
def __init__(self, csv_path, indexes):
    # initilizes some variables
    self.csv_path = csv_path
    self.norm_facor = 255.0
    self.indexes = indexes
    random.shuffle(self.indexes)
```

- We save the csv_path input as a class variable, in order to be used in other functions.
- indexes indicate which lines of the csv file will be used in the dataset.
- We also define two other variables which will be used later.
- You can conduct all initialization processing before actually fetching the sample from dataset.

Loading Datasets

Defining a data generator

The data generator is implemented as a class, which has three key methods: __init__, __len__, and __getitem__.

```
def __len__(self):
    # return the total number of sa
mples in the dataset
    return len(self.indexes)
```

- The function returns the total number of samples in the dataset
- Given this value the data generator knows whether all the data has been traversed. The “index” parameter in __getitem__ ranges in [0, returned value-1].
- In this example, since `self.indexes` stores the line indexes, we use the length of the list to indicate the number of samples.

Loading Datasets

Defining a data generator

The data generator is implemented as a class, which has three key methods: __init__, __len__, and __getitem__.

```
def __getitem__(self,index):
    # get one sample according to the index
    line_index = self.indexes[index]
    line_str = linecache.getline(self.csv_path,line_index)
    line_val = [int(i) for i in line_str.split(',')]
    label = line_val[0]
    feat = np.array(line_val[1:])/self.norm_facor
    return feat,label
```

- The index value will iterate from 0 to L-1, where L is the value returned by __len__;
- __getitem__ function defines the rules of creating the sample given the index
- Here, the rules are
  1. Determine which line should be read
  2. Read the content of the chosen line to a string
  3. Convert the string to a list containing integer elements
  4. Get the feature and label
  5. return the values

Loading Datasets

Using a data generator

Once the generator class is defined, the data generator instance can be further created

```
indexes = [i for i in range(60000)]
train_index = indexes[6000:]
val_index = indexes[:6000]
train_set = DataGenerator('mnist_train.csv',train_index)
val_set = DataGenerator('mnist_train.csv',val_index)
print(len(train_set))
print(len(val_set))

cnt = 0
for x, y in train_set:
  print(y)
  cnt = cnt+1
  if cnt>=5:
    break
```

The file directory and indexes are used as input parameters

```
54000
6000
7
1
7
1
6
```

Try run the codes for multiple times and check the outputs. Why each run gives a different output?

Loading Datasets

Defining the learning target

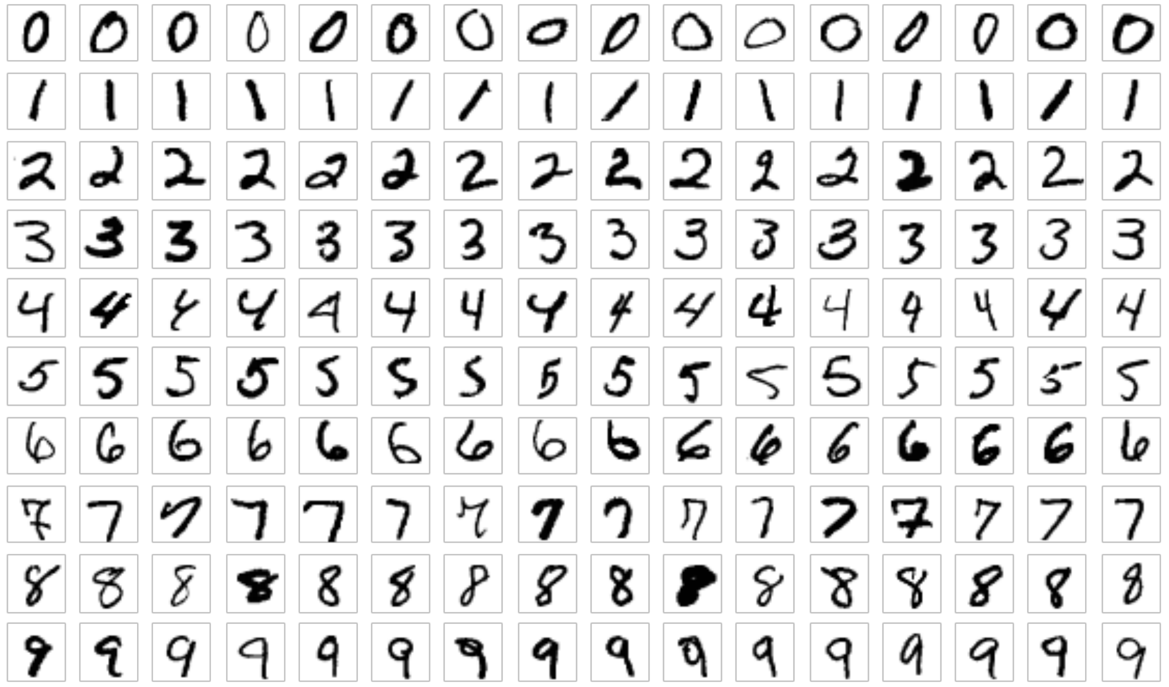
The learning target is defined as the label when creating a sample.

```python
def __getitem__(self,index):
    # get one sample according to the index
    line_index = self.indexes[index]
    line_str = linecache.getline(self.csv_path,line_index)
    line_val = [int(i) for i in line_str.split(',')]
    label = line_val[0]
    feat = np.array(line_val[1:])/self.norm_facor
    return feat,label
```

In our case, we want to classify the digits, therefore, the digit value itself, stored in the first element in each line, is taken as the learning target

Exercise

1. Rather than identifying the actual number, we want to classify whether the digit image contains an odd number. Modify the data generator to produce correct data samples from the MNIST dataset.



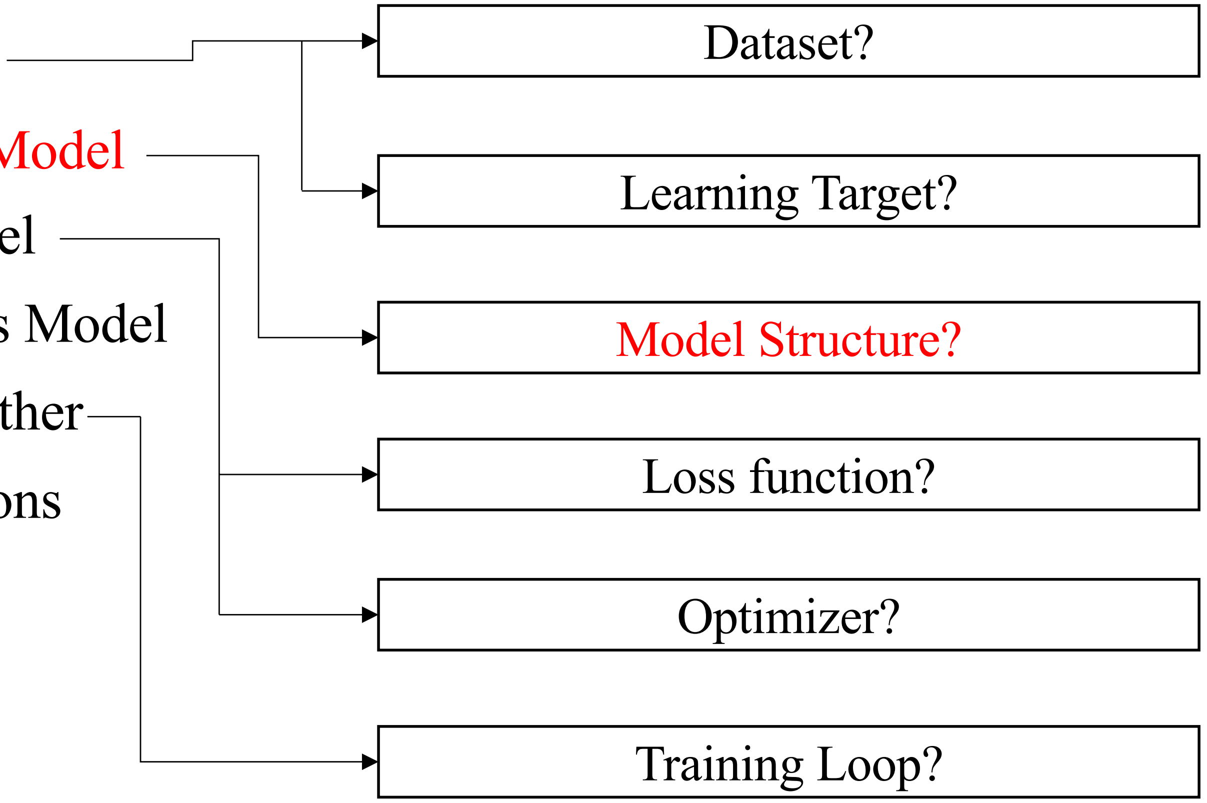
0
1
0
1
0
1
0
1
0
1

Exercise

1. Rather than identifying the actual number, we want to classify whether the digit image contains an odd number. Modify the data generator to produce correct data samples from the MNIST dataset.
2. We want to make the input feature normalized to [-1,1]. What shall we do?

Key Components of Keras Pipeline

1. Load Data
2. Define Keras Model
3. Fit Keras Model
4. Evaluate Keras Model
5. Tie It All Together
6. Make Predictions

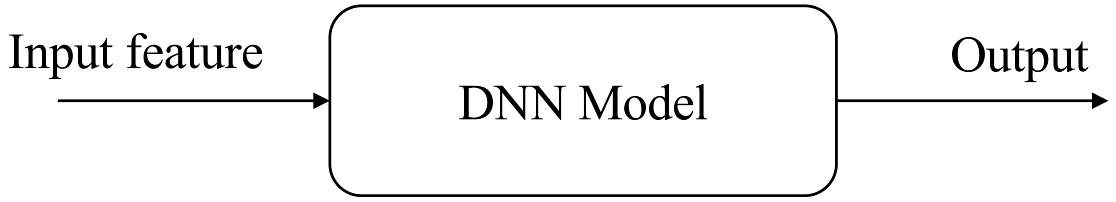


Define Keras Model

A model is the core of deep learning, is composed of many building blocks and defines the computation logics from input to output.

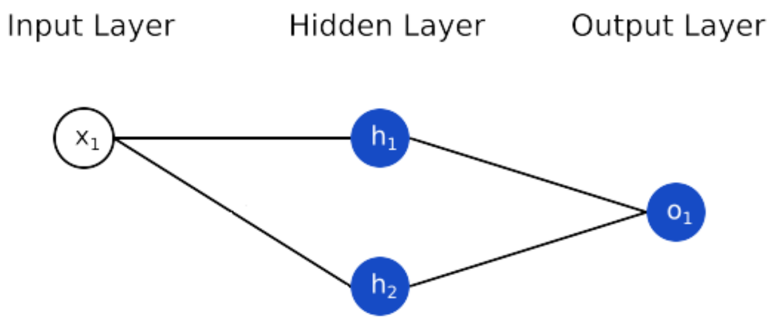
Two different ways of defining a deep neural network:

- **Keras' Sequential API**
- Keras' Functional API

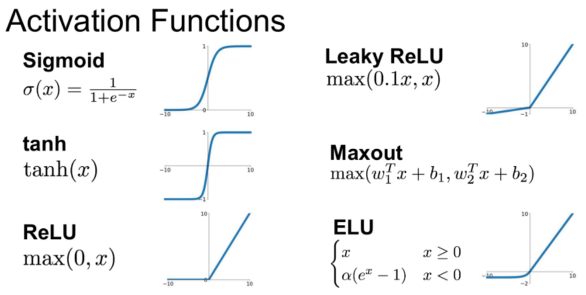


Define Keras Model

Keras' Sequential API



DNN structure



Activation

Define a simple fully connected network with linear layers

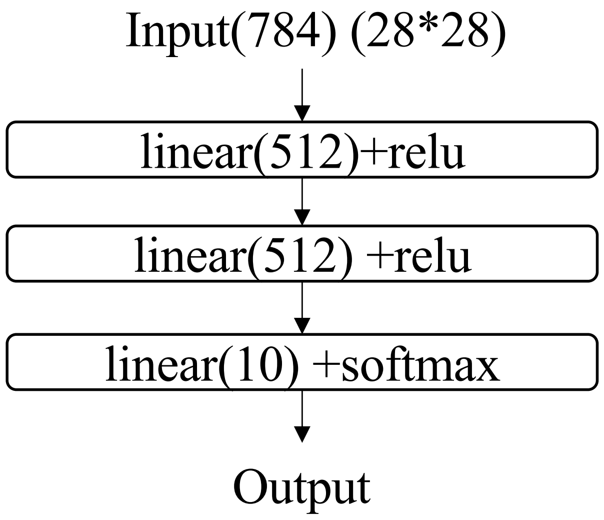
```
# Define the model:
from keras.models import Sequential
from keras.layers import Dense
model = Sequential()
model.add(Dense(2, input_dim=1, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

Input & hidden layer, 2 nodes in the hidden layer

Output layer, one node in the output layer

Define Keras Model

Keras' Sequential API

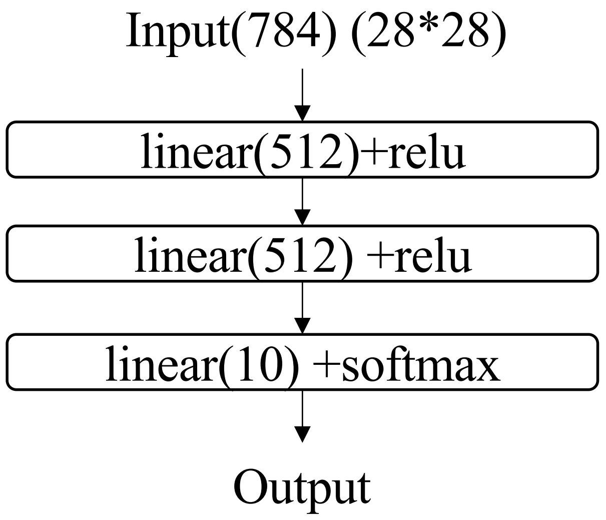


```
# Define the model:
from keras.models import Sequential
from keras.layers import Dense
model = Sequential()
model.add(Dense(512, input_shape=(784,), activation='relu'))

model.add(Dense(512, activation='relu'))
model.add(Dense(10, activation='softmax'))
model.summary()
```

Now we define a fully connected network for MNIST inputs
The Dense layer can automatically figure out the input dimension when used as a hidden layer.

Define Keras Model

Keras' Sequential API

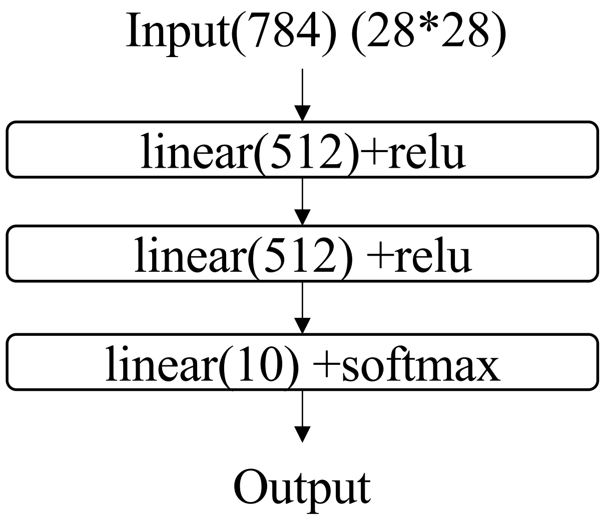


```
# Define the model:
from keras.models import Sequential
from keras.layers import Dense,Input
model = Sequential()
model.add(Input(shape=(784,)))
model.add(Dense(512, activation='relu'))
model.add(Dense(512, activation='relu'))
model.add(Dense(10, activation='softmax'))
model.summary()
```

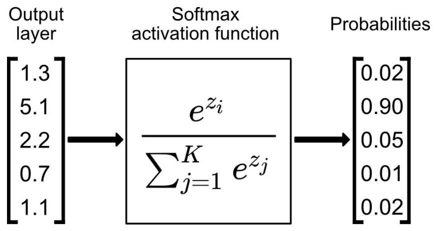
We can also use a separate "input layer" to specify the input dimension

Define Keras Model

Keras' Sequential API



```
# Define the model:
from keras.models import Sequential
from keras.layers import Dense,Input
model = Sequential()
model.add(Input(shape=(784,)))
model.add(Dense(512, activation='relu'))
model.add(Dense(512, activation='relu'))
model.add(Dense(10, activation='softmax'))
model.summary()
```


Output layer
1.3
5.1
2.2
0.7
1.1
Softmax activation function
$\frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$
Probabilities
0.02
0.90
0.05
0.01
0.02

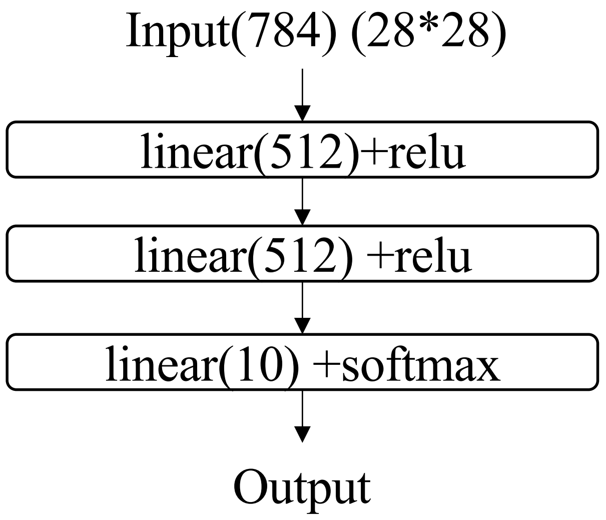
- The activation gives nonlinearity to the model, popular activation functions include **sigmoid, relu and softmax**.
- Softmax normalizes a vector to ensure the summation is one, which is suitable to indicate the probability of each class.

DEPARTMENT OF COMPUTER SCIENCE 計算機科學系

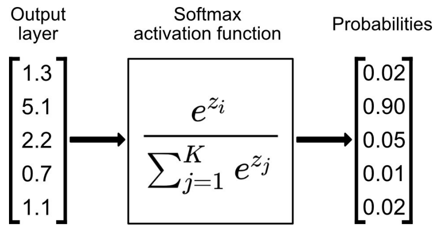
香港浸會大學 HONG KONG BAPTIST UNIVERSITY

Define Keras Model

Keras' Sequential API



```
# Define the model:
from keras.models import Sequential
from keras.layers import Dense,Input
model = Sequential()
model.add(Input(shape=(784,)))
model.add(Dense(512, activation='relu'))
model.add(Dense(512, activation='relu'))
model.add(Dense(10, activation='softmax'))
model.summary()
```

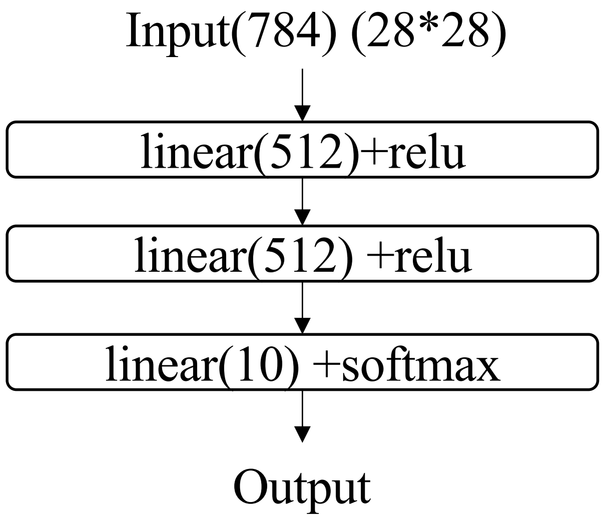


Output layer
1.3
5.1
2.2
0.7
1.1
Softmax activation function
$\frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$
Probabilities
0.02
0.90
0.05
0.01
0.02

- To understand **sigmoid, relu and softmax** activations, please see https://en.wikipedia.org/wiki/Activation_function

Define Keras Model

Keras' Sequential API

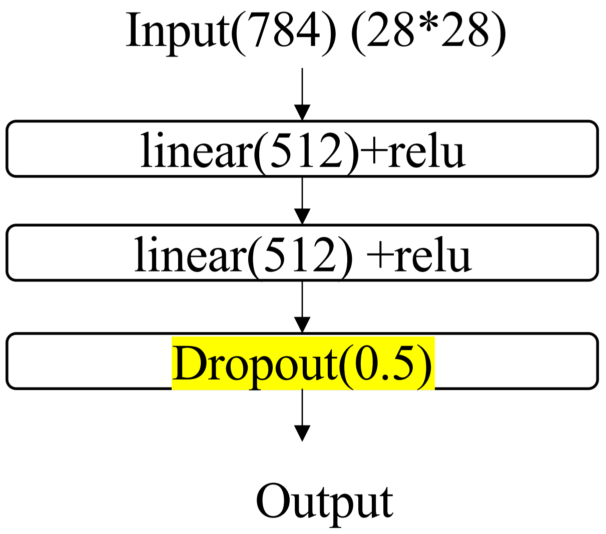


```
# Define the model:
from keras.models import Sequential
from keras.layers import Dense,Input
model = Sequential(
     [Input(shape=(784,)),
     Dense(512, activation='relu'),
     Dense(512, activation='relu'),
     Dense(10, activation='softmax')]
     )
model.summary()
```

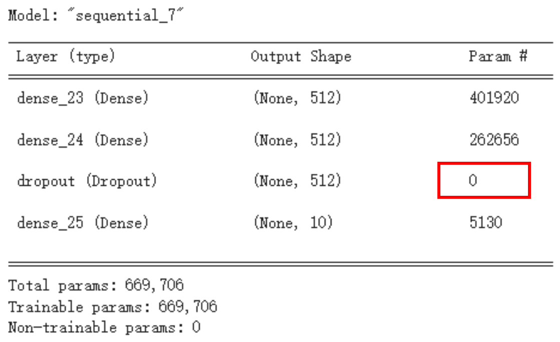
- The Sequential API can also take as input a list of layers.

Define Keras Model

Keras' Sequential API



```
# Define the model using functional API:
from keras.models import Sequential
from keras.layers import Dense,Input,Dropout

x = Input(shape=(784,))

model = Sequential(
     [Input(shape=(784,)),
     Dense(512, activation='relu'),
     Dense(512, activation='relu'),
     Dropout(0.5),
     Dense(10, activation='softmax')]
     )
model.summary()
```



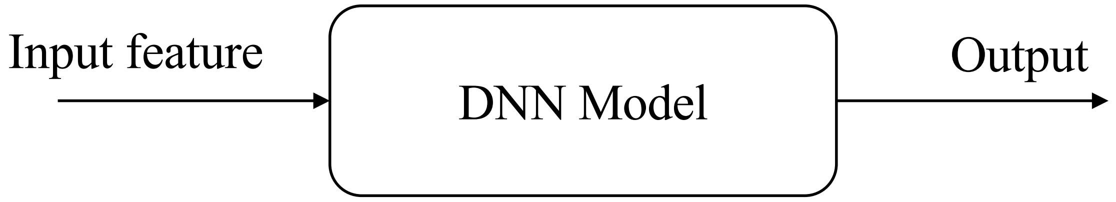
```
Model: "sequential_7"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_23 (Dense)            (None, 512)               401920

 dense_24 (Dense)            (None, 512)               262656

 dropout (Dropout)           (None, 512)               0

 dense_25 (Dense)            (None, 10)                5130

=================================================================
Total params: 669,706
Trainable params: 669,706
Non-trainable params: 0
```

- A dropout layer is usually used to prevent overfitting.
- It randomly drop neurons during training, the rate of dropout changes from 0 to 1.

Define Keras Model

A model is the core of deep learning, is composed of many building blocks and defines the computation logics from input to output.

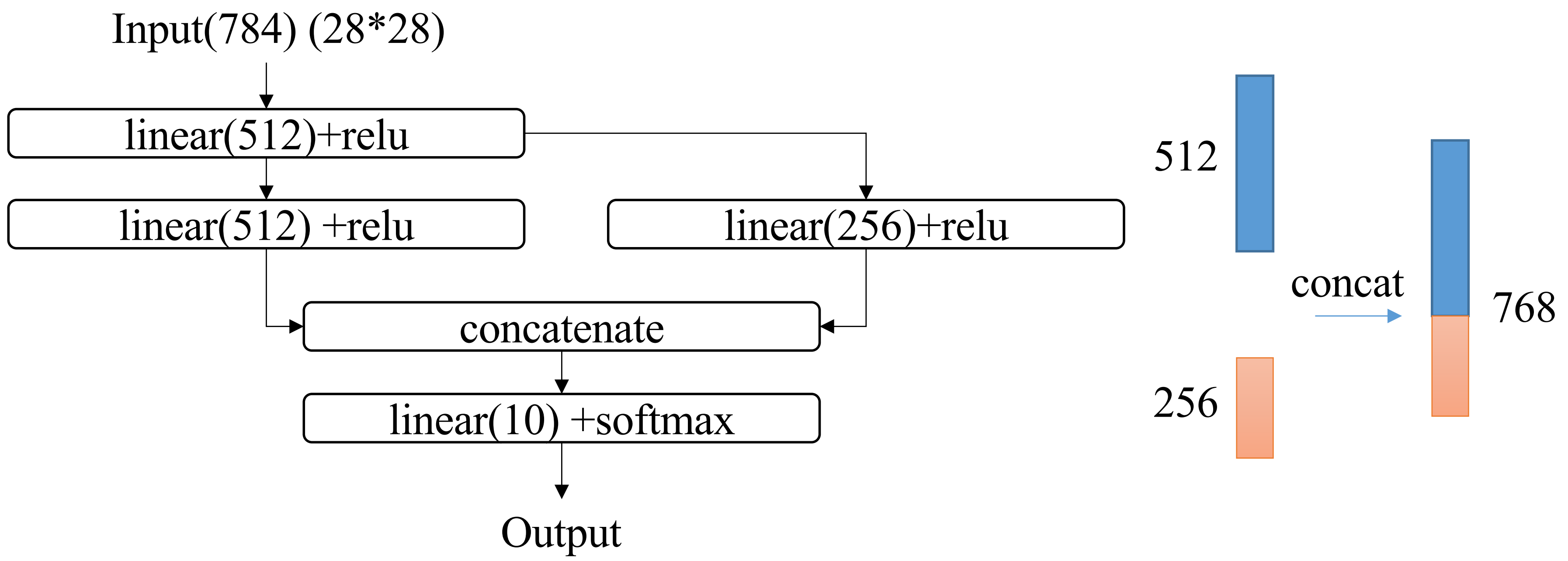
- Two different ways of defining a deep neural network:
- Keras' Sequential API
- **Keras' Functional API**



Define Keras Model

Keras' Sequential API

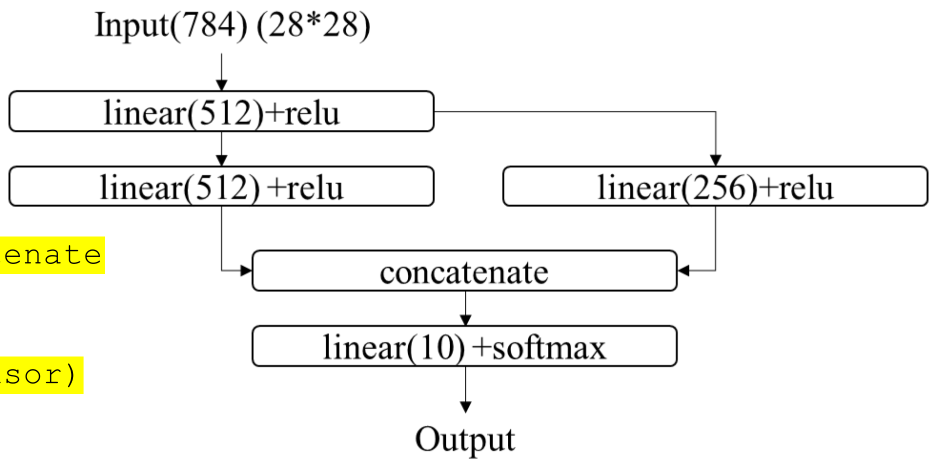
- In many cases, the model structure is more complicated than forward connections.



Define Keras Model

Keras' Sequential API

```python
# Define the model using functional API:
from keras.models import Model
from keras.layers import Dense,Input,concatenate

input_tensor = Input(shape=(784,))
x = Dense(512, activation='relu')(input_tensor)
x1 = Dense(512, activation='relu')(x)
x2 = Dense(256, activation='relu')(x)
x3 = concatenate([x1,x2])
output_tensor = Dense(10, activation='softmax')(x3)
model = Model(inputs=input_tensor, outputs=output_tensor)
model.summary()
```



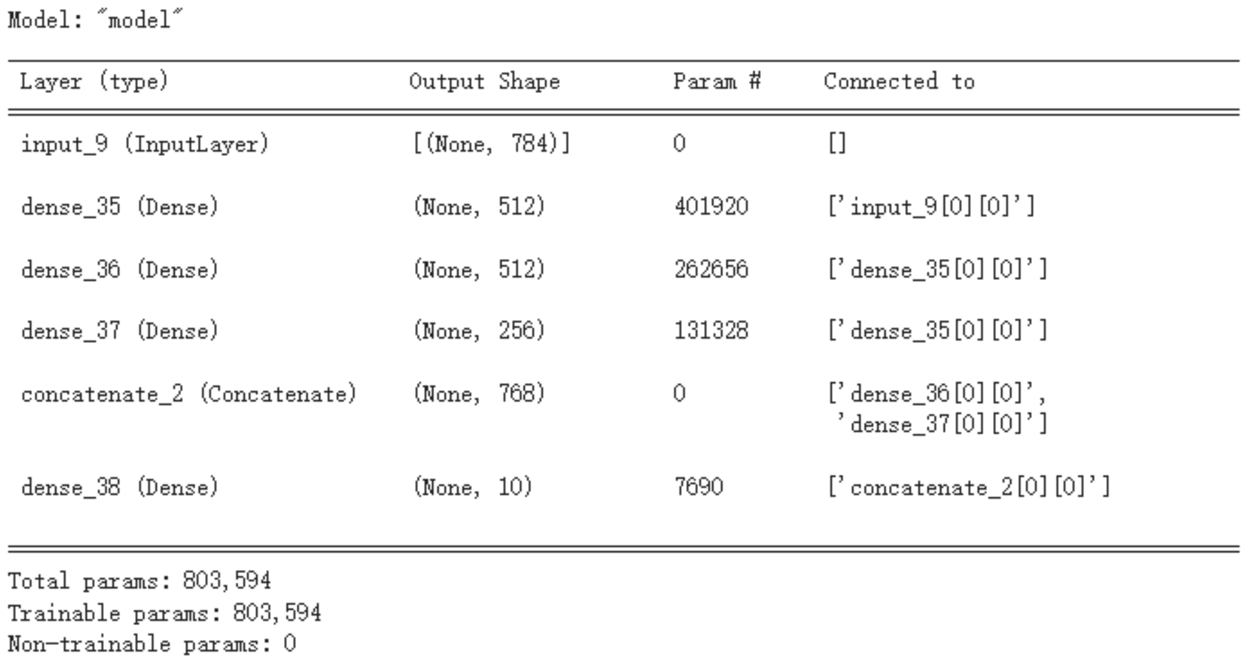
"concatenate" rather than "Concatenate"

Learn the syntax of forwarding the inputs

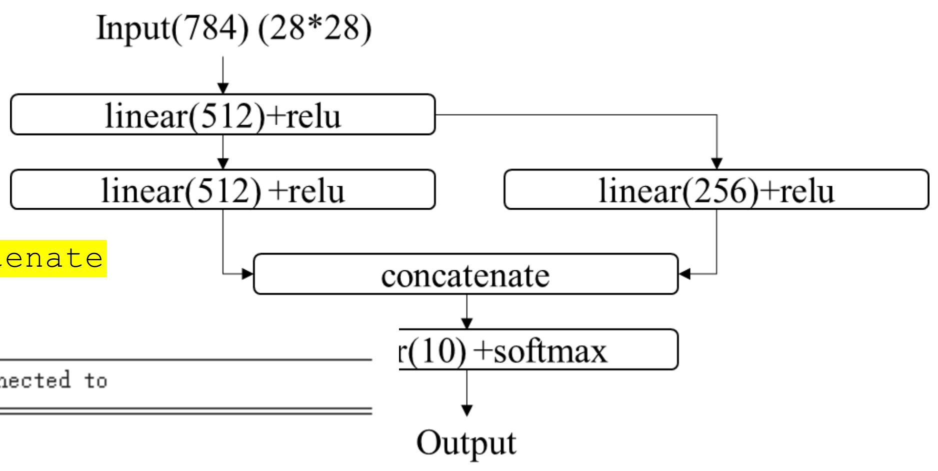
Define Keras Model

Keras’ Sequential API

```python
# Define the model using functional API:
from keras.models import Model
from keras.layers import Dense,Input,concatenate
```



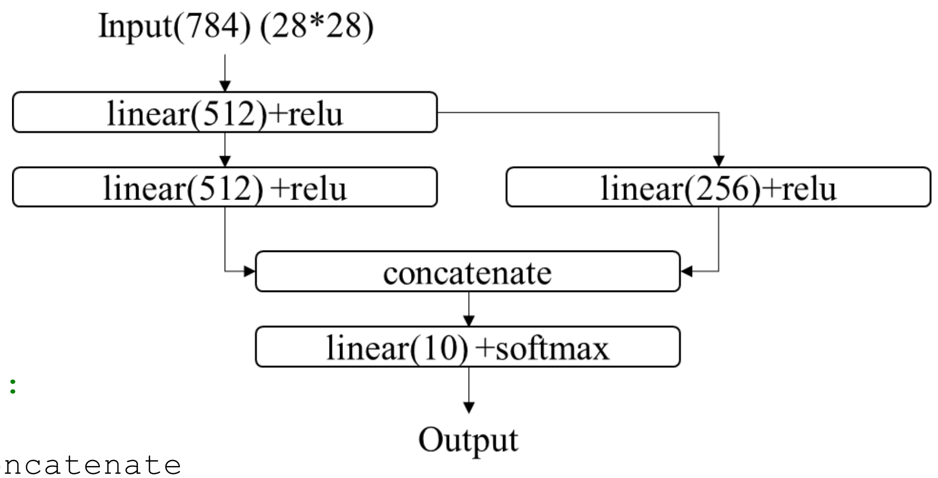
```
Model: "model"
_____________________________________________________________________________
Layer (type)                Output Shape      Param #    Connected to
=============================================================================
input_9 (InputLayer)        [(None, 784)]     0          []

dense_35 (Dense)            (None, 512)       401920     ['input_9[0][0]']

dense_36 (Dense)            (None, 512)       262656     ['dense_35[0][0]']

dense_37 (Dense)            (None, 256)       131328     ['dense_35[0][0]']

concatenate_2 (Concatenate) (None, 768)       0          ['dense_36[0][0]',
                                                          'dense_37[0][0]']

dense_38 (Dense)            (None, 10)        7690       ['concatenate_2[0][0]']

=============================================================================
Total params: 803,594
Trainable params: 803,594
Non-trainable params: 0
```



r than “Concatenate”

forwarding the inputs

Define Keras Model



Keras' Sequential API

Functional API makes it possible to "package" a model as a function

```
# Define the model using functional API:
from keras.models import Model
from keras.layers import Dense,Input,concatenate
```

```
def MNIST_Model(in_shape,out_shape):
  input_tensor = Input(shape=(in_shape,))
  x = Dense(512, activation='relu')(input_tensor)
  x1 = Dense(512, activation='relu')(x)
  x2 = Dense(256, activation='relu')(x)
  x3 = concatenate([x1,x2])
  output_tensor = Dense(out_shape, activation='softmax')(x3)
  model = Model(inputs=input_tensor, outputs=output_tensor)
  return model
```

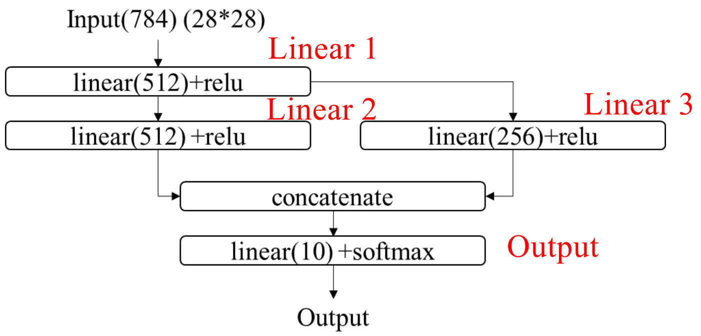
```
my_model = MNIST_Model(784,10)
my_model.summary()
```

We can easily change the hyperparameters!

Define Keras Model

Keras' Sequential API

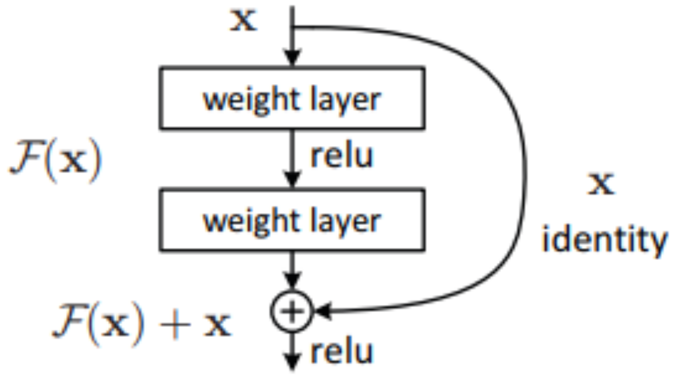
We suggest a clearer way of defining the model:
Define layers, operations, models.
<u>The operations are easier to read when giving layer names in the codes.</u>



```
# Define the model using functional API:
from keras.models import Model
from keras.layers import Dense,Input,concatenate

def MNIST_Model(in_shape,out_shape):
  # define layers
  input_tensor = Input(shape=(in_shape,))
  linear_layer1 = Dense(512, activation='relu')
  linear_layer2 = Dense(512, activation='relu')
  linear_layer3 = Dense(512, activation='relu')
  out_layer = Dense(out_shape, activation='softmax')

  # define operations
  x = linear_layer1(input_tensor)
  x1 = linear_layer2(x)
  x2 = linear_layer3(x)
  x3 = concatenate([x1,x2])
  output_tensor = out_layer(x3)

  # define the model
  model = Model(inputs=input_tensor, outputs=output_tensor)
  return model

my_model = MNIST_Model(784,10)
my_model.summary()
```

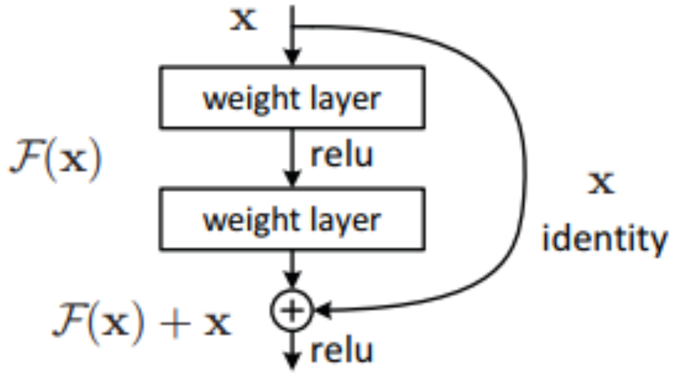
Exercise

1. Rather than identifying the actual number, we want to classify whether the digit image contains an odd number. Modify the data generator to produce correct data samples from the MNIST dataset.
2. We want to make the input feature normalized to [-1,1]. What shall we do?
3. Define the resnet structure with all weight layers as 256-node dense layers.


x
weight layer
relu
weight layer
$\mathcal{F}(\mathbf{x})$
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

Exercise

1. Rather than identifying the actual number, we want to classify whether the digit image contains an odd number. Modify the data generator to produce correct data samples from the MNIST dataset.
2. We want to make the input feature normalized to [-1,1]. What shall we do?
3. Define a resnet block with all weight layers as 256-node dense layers.
4. Define a large resnet consisting of 80 resnet blocks.



x
weight layer
$\mathcal{F}(\mathbf{x})$
relu
weight layer
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu