

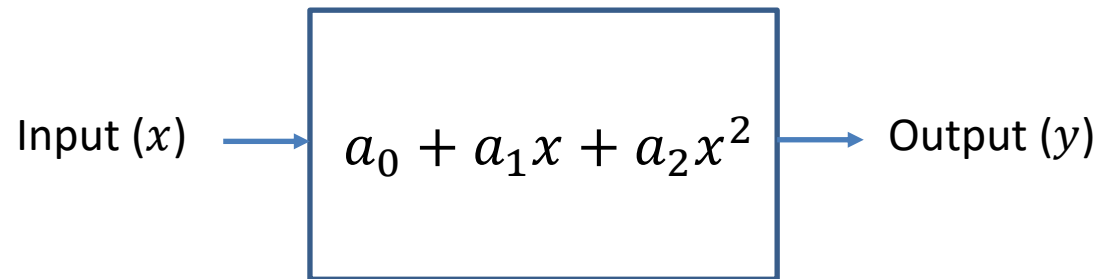
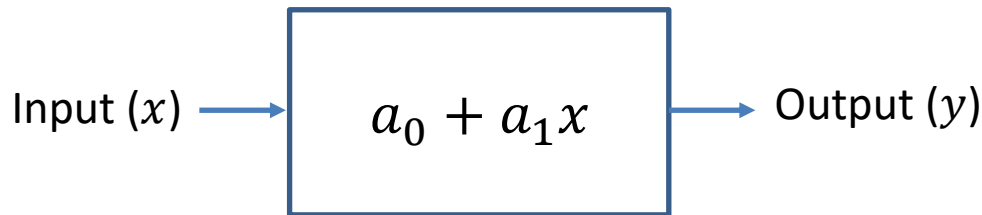
Keras and TensorFlow



Acknowledgement: Un-copyrighted photos and pictures were taken from the Internet

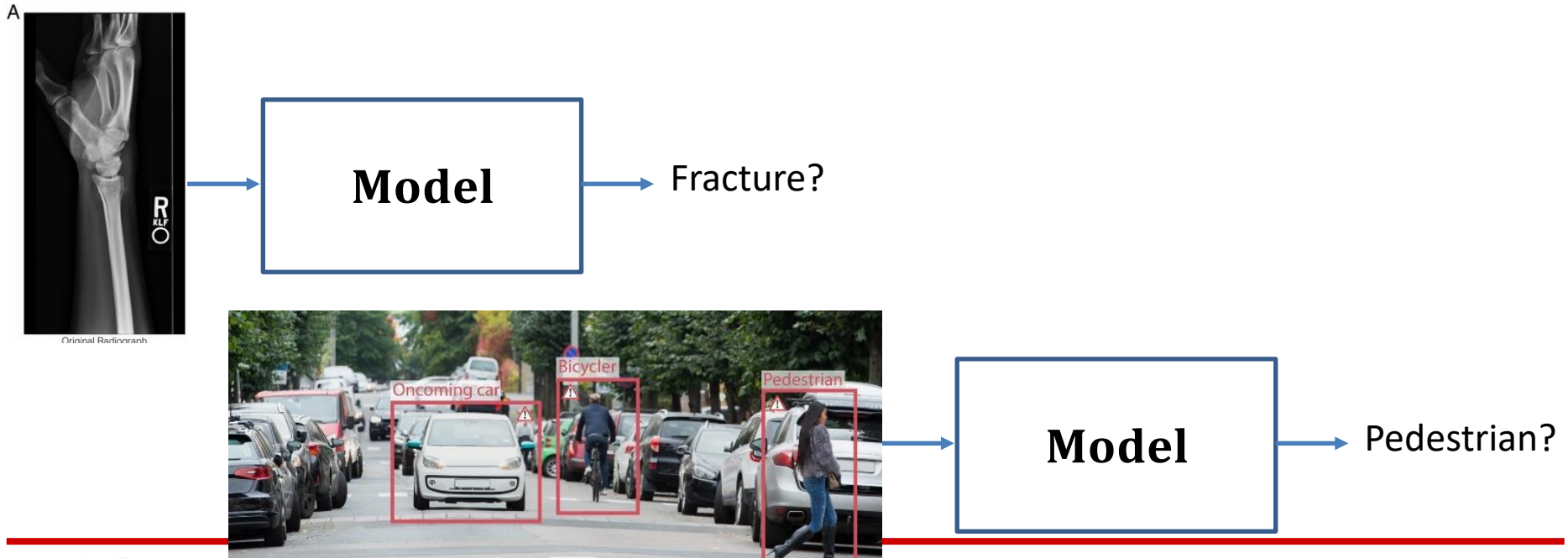
Motivation: Neural Networks

- So far, you have learned how to fit a **simple model** into your data.
- For example, we can easily fit **linear or quadratic models** to our data.
- We can **easily find** the **parameters** of these **models** using **data**.



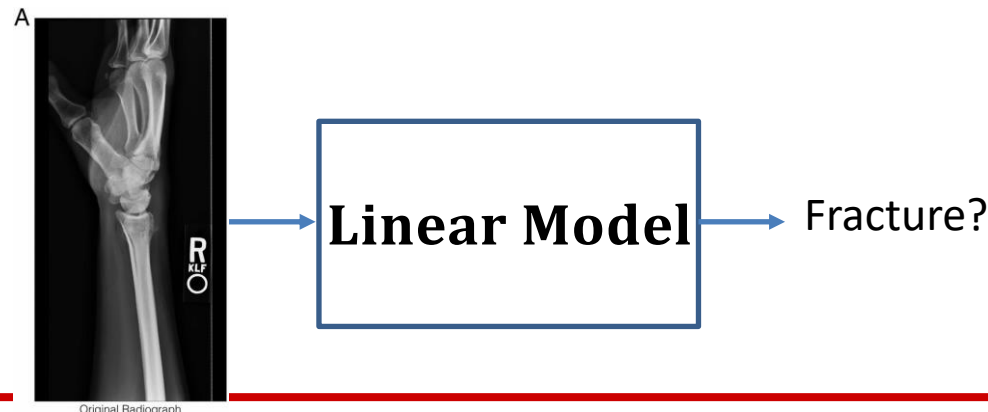
Motivation: Neural Networks

- Simple models may not be helpful for certain applications.
- Autonomous driving: Is there a pedestrian on the road?
- Bone fracture detection: Is this bone broken?



Motivation: Neural Networks

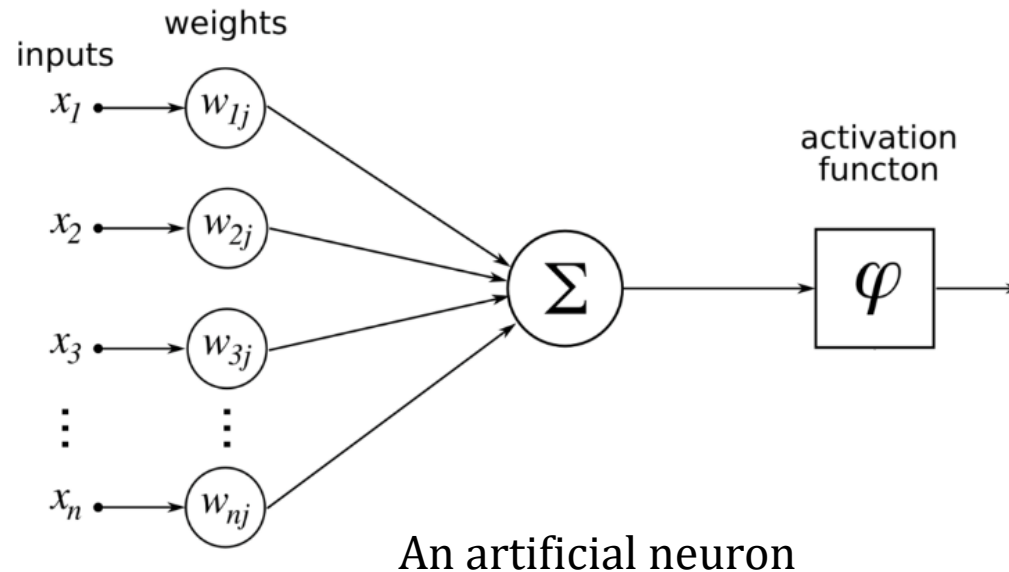
- **Autonomous driving:** Is there a pedestrian on the road?
- **Bone fracture detection:** Is this bone broken?
- In such cases, simple models are not helpful.
- Simple models **may not** provide **accurate predictions** in these cases.



Motivation: Neural Networks

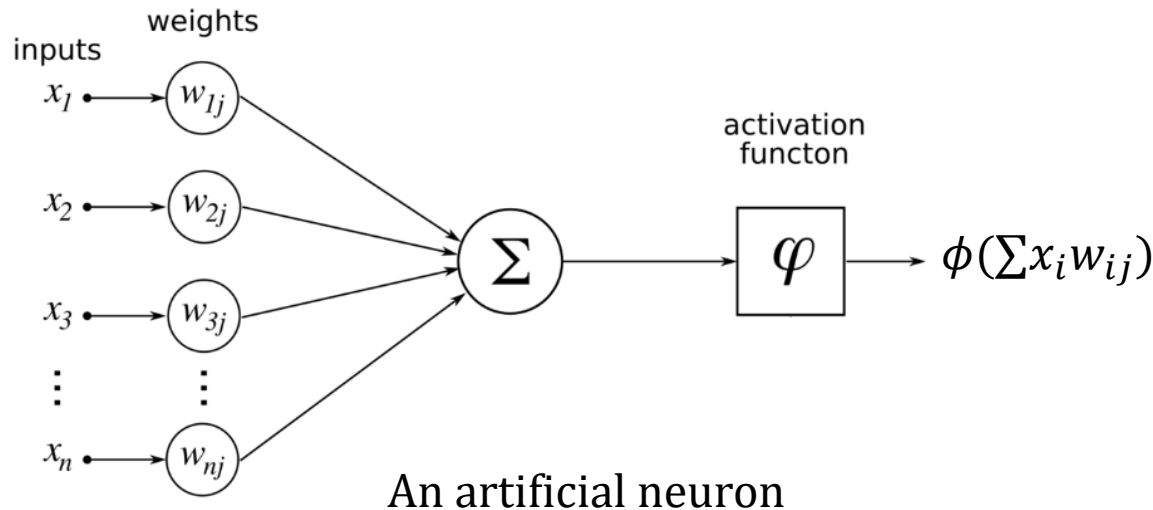
- Neural networks allow us to build sophisticated models.
- Artificial neurons are essential building blocks of neural networks.
- Each artificial neuron may have multiple inputs and outputs.

Inputs are numbers.



Motivation: Neural Networks

- Each neuron **combines** its **inputs** using a **set of weights**.
- It then **passes** the result **through** an **activation function**.
- The **output** of a **neuron might** be the **input** of other neurons.

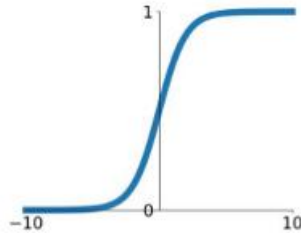


Motivation: Neural Networks

- Examples of activation functions.

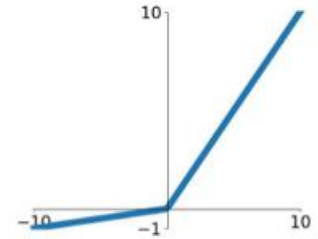
Sigmoid

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



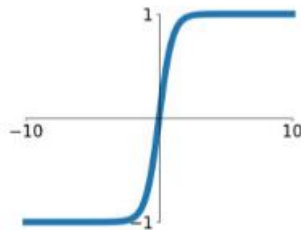
Leaky ReLU

$$\max(0.1x, x)$$



tanh

$$\tanh(x)$$

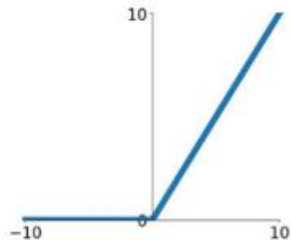


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

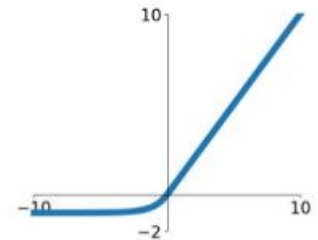
ReLU

$$\max(0, x)$$

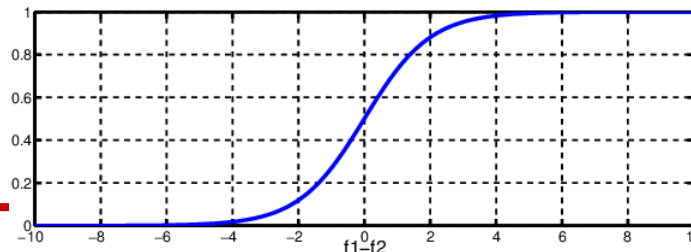


ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



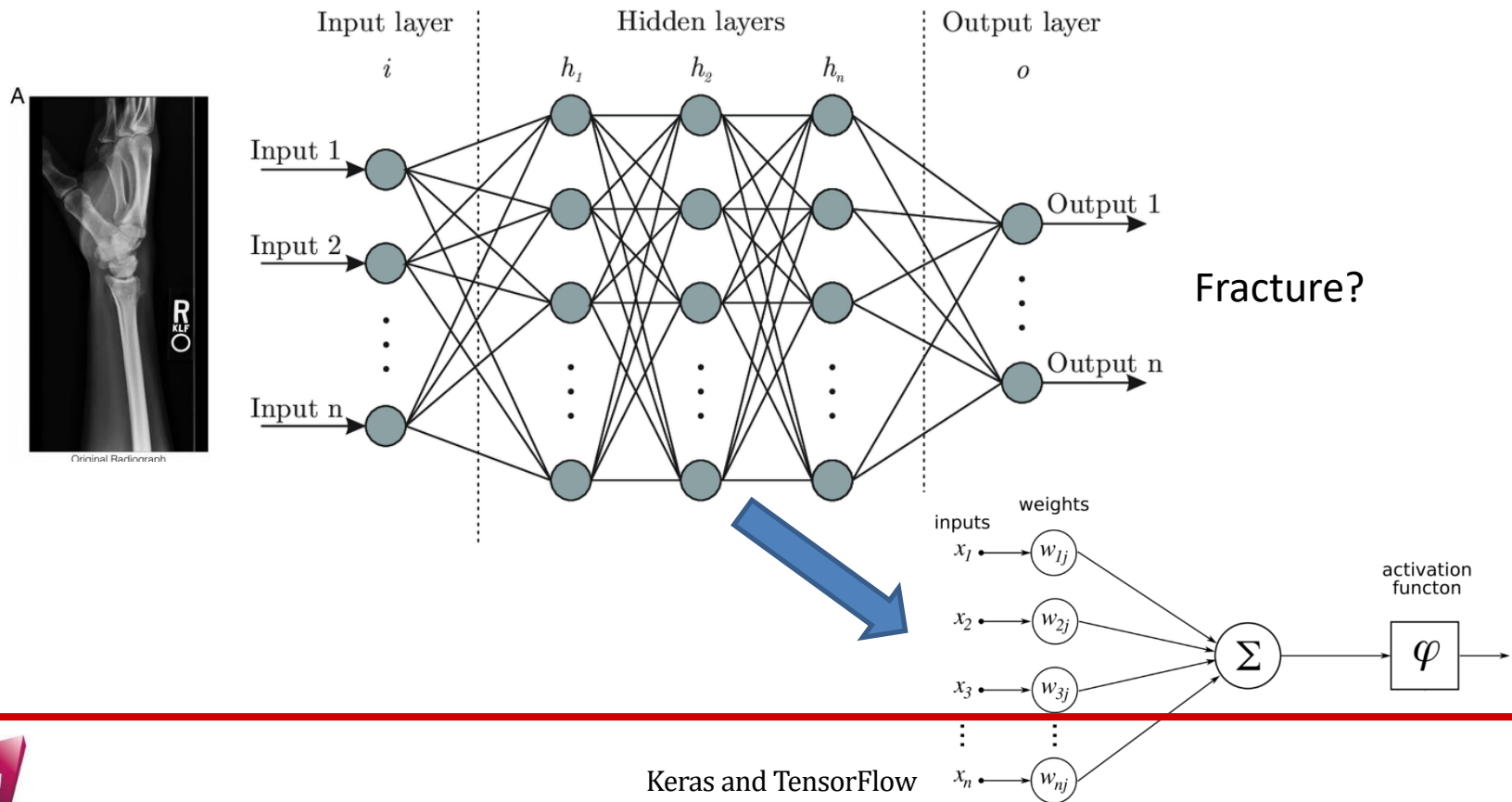
Softmax



Keras and TensorFlow

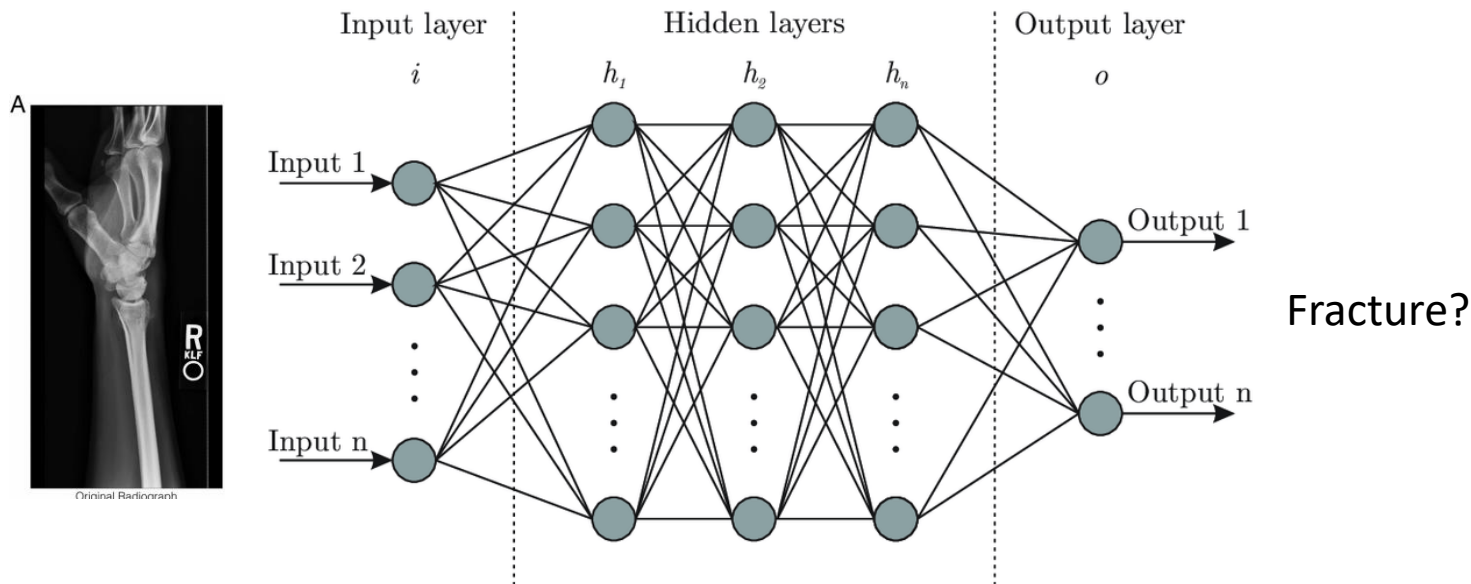
Motivation: Neural Networks

- An **artificial neural** network is **built** by **connecting** a set of **neurons**.
- We have an **input layer**, an **output layer**, and hidden layers.



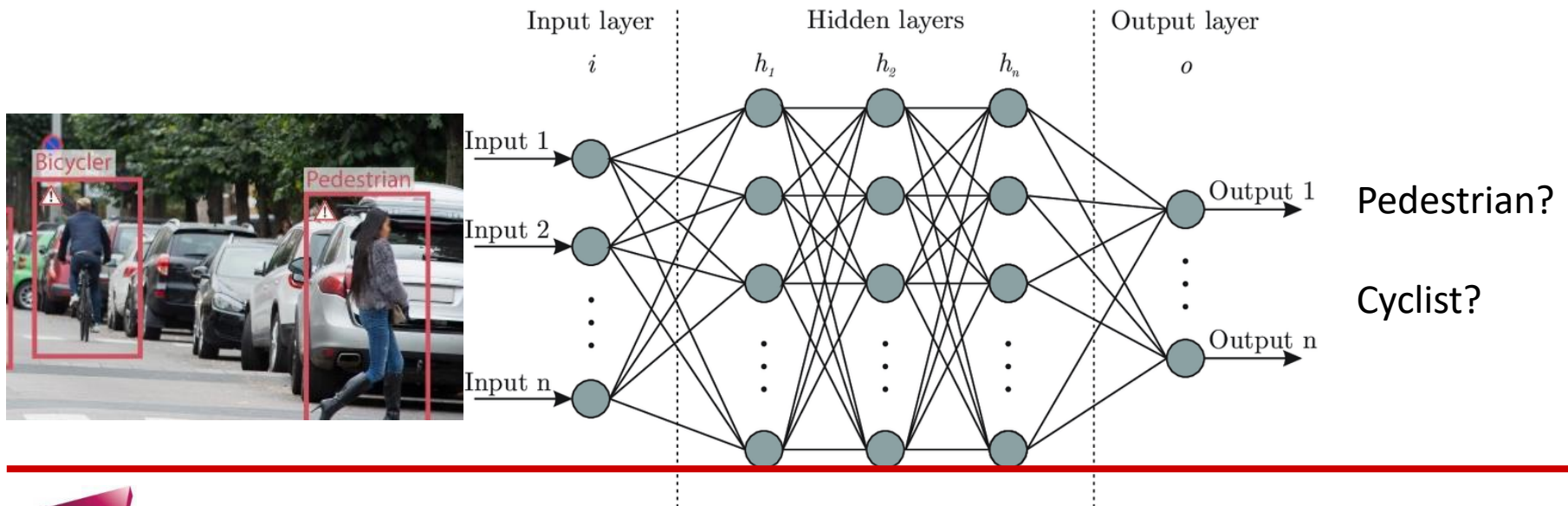
Motivation: Neural Networks

- The **input data** is fed to the **input layer**.
- The **output layer** **generates** the **prediction**.



Motivation: Neural Networks

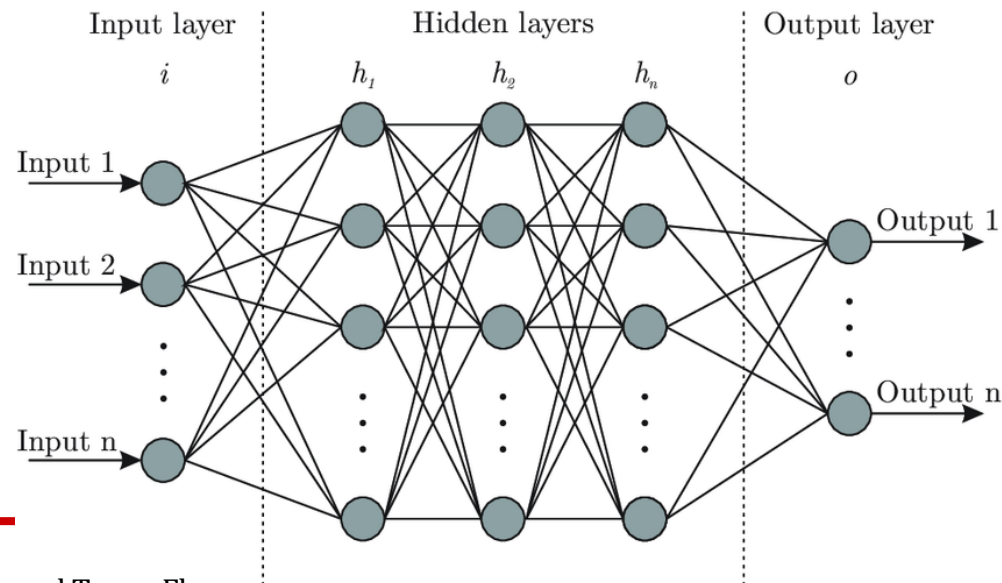
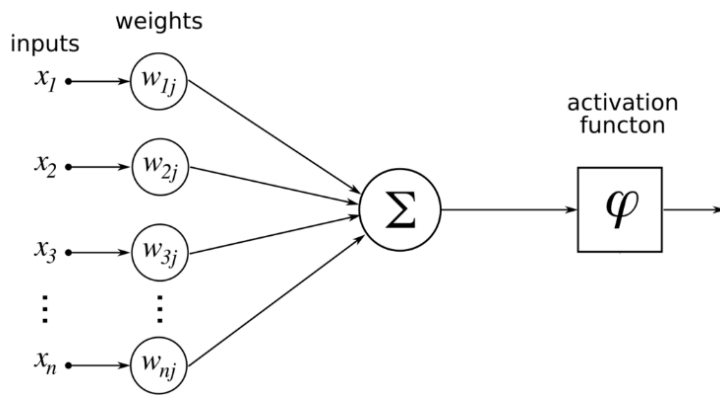
- Suppose that we need to build an **artificial neural network** for an autonomous driving application.
- The **input** to the **network** is an **image**.
- The **output** maybe the **prediction** of the pedestrians in the image.



Keras and TensorFlow

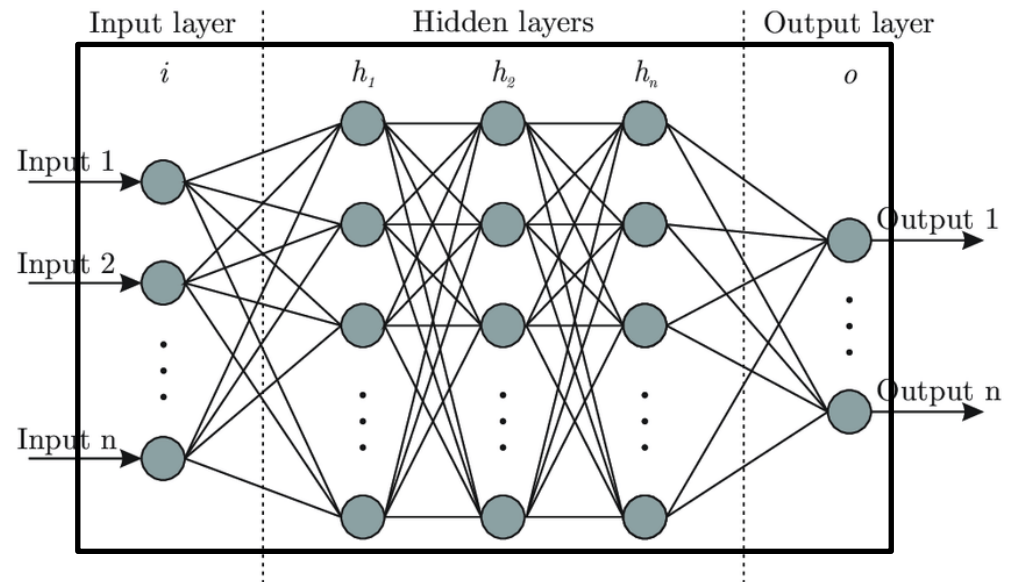
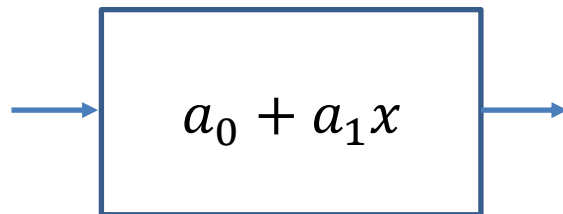
Motivation: Neural Networks

- In order to build this model, we need to find the **weights** using **data**.
- We may have **thousands of neurons**. Each neuron might have **hundreds** of weights.
- It is **difficult** to find the **weights** by hand.



Motivation: Neural Networks

- Simple models have formulas for computing the parameters from data.
- For complicated models, there are no formulas for computing weights.
- We need to find the weights using differentiation.



Neural Networks in Python

- Manually implementation of complex neural networks in Python is cumbersome.
- We may not be able to easily use an implementation for different applications.
- TensorFlow and Keras allow you to implement neural networks in Python without getting involved in low-level implementation.



TensorFlow

- TensorFlow is a free and open-source software developed by Google.
- It can efficiently compute the weights of a huge neural network.
- TensorFlow can be installed and used in Python.



TensorFlow

Keras

- Keras is open-source software that provides an interface for neural networks in Python.
- Keras is included in TensorFlow.
- Use the following commands to install TensorFlow in your Jupyter notebook

```
from pip._internal import main  
main(['install','tensorflow'])
```



Workflow

Import Data (input, output components)



Specify the model and its parameters



Fit the model to data

TensorFlow and Keras

- To use **TensorFlow and Keras**, you need to **import them** in your Python code.

```
import tensorflow as tf
```

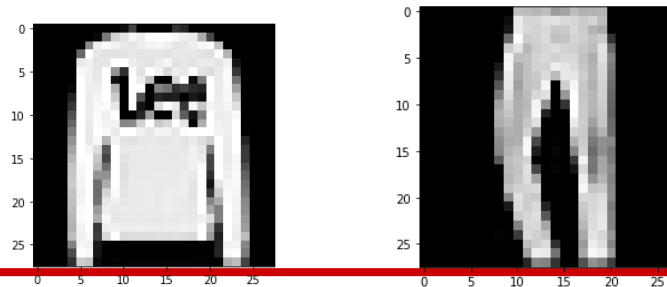
```
from tensorflow import keras
```

- **TensorFlow** has many **datasets** that can be used **to build models**.
- You can check the available datasets at the following link
<https://www.tensorflow.org/datasets/catalog/overview>
- In this lecture, we use a few datasets to build models using **TensorFlow**.

Example: Object Identification

Example

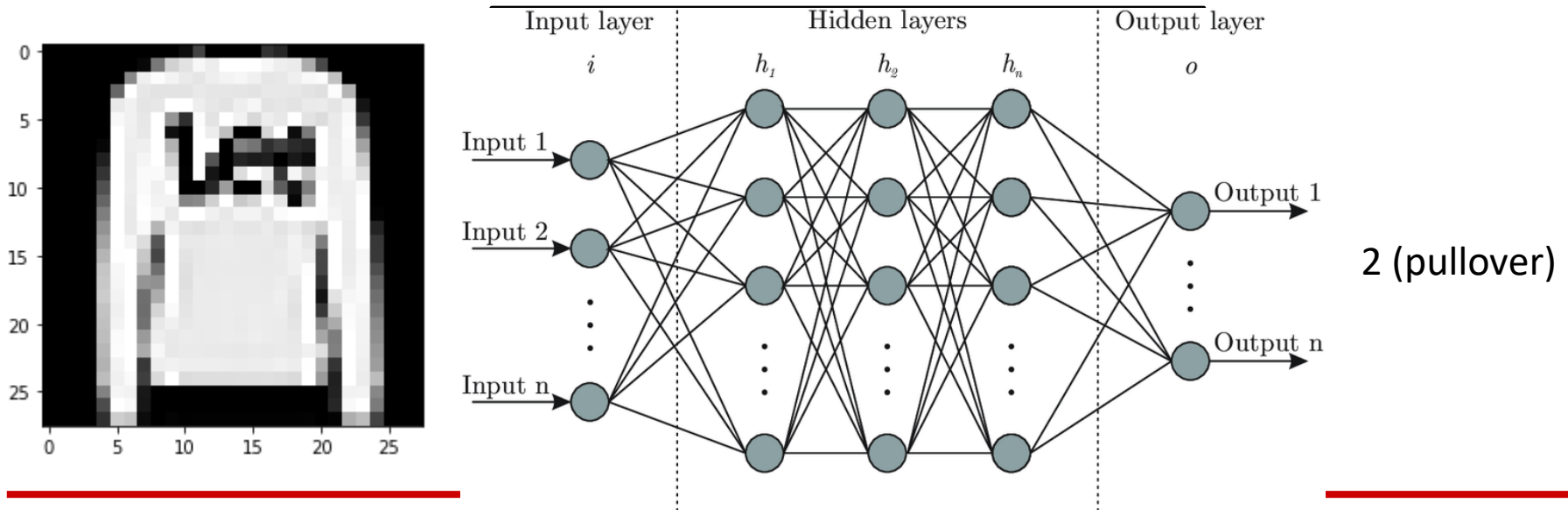
- **fashion_mnist** is a **dataset** that **contains images** of **items** such **shirt, coat, etc.**
- Each **image** in the dataset **belongs** to one of the **following categories.**
- Each **category** is **labeled** by a **number.**
- 1: Trouser
- 2: Pullover



Label	Description
0	↔ T-shirt/top
1	↔ Trouser
2	↔ Pullover
3	↔ Dress
4	↔ Coat
5	↔ Sandal
6	↔ Shirt
7	↔ Sneaker
8	↔ Bag
9	↔ Ankle boot

Example

- We would like to build a **model** to **predict the label** of an image.
- This allows us to **identify** items using **images**.
- We do not need a **person** to **identify the objects**. A **computer program** can identify them.



Example

- You can use the **following commands** to load the **fashion_mnist** dataset and use it in your code.

```
fashion_mnist = keras.datasets.fashion_mnist
```

```
(X_train_full, y_train_full), (X_test, y_test) = fashion_mnist.load_data()
```

Training Data

Test Data

- X_train_full** and **X_test** contain a set of images.
- y_train_full** contains the label of each image in **X_train_full**.
- y_test** contains the label of each image in **X_test**.

Example

- Let's look at some of these data

```
pyplot.imshow(X_train_full[1], cmap=pyplot.get_cmap('gray'))  
print(y_train_full[1])
```

```
pyplot.imshow(X_test[1], cmap=pyplot.get_cmap('gray'))  
print(y_test[1])
```

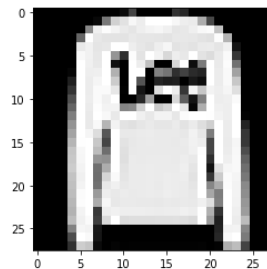
Training Dataset

- Training dataset is used to find the parameters of the neural network.
- The training dataset has an input component and an output component.

Training Data

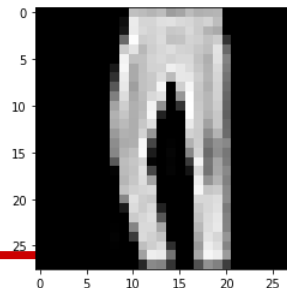
→ (X_train_full, y_train_full)

Input



Output

2 (pullover)



1 (trouser)

Label	Description
0	T-shirt/top
1	Trouser
2	Pullover
3	Dress
4	Coat
5	Sandal
6	Shirt
7	Sneaker
8	Bag
9	Ankle boot

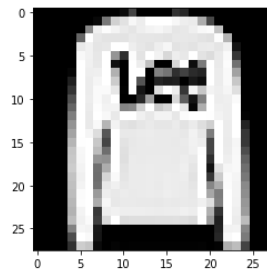
Training Dataset

- There are 60000 images in the training set.
- Each image is presented by an **array** of **28-by-28**. The array **contains** the **intensity** of **each pixel**.

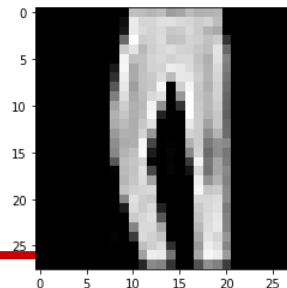
Training Data $\longrightarrow$ (X_train_full, y_train_full)

Input

Output



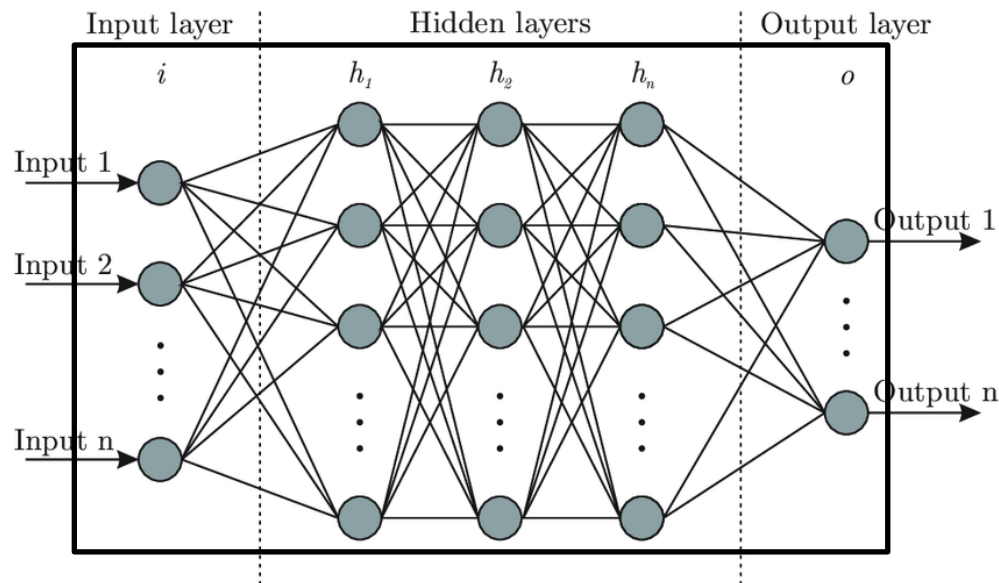
2 (pullover)



1 (trouser)

Motivation: Neural Networks

- In order to build this model, we need to find the **weights** using **data points**.
- Each data point has an **input** and an **output** component.
- We can predict the output for each image.

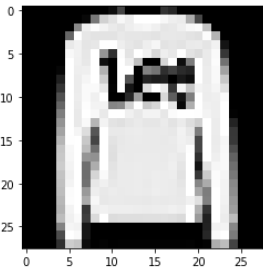


Test Dataset

- **Test dataset** is used to **check the performance** of the neural network that we obtain in the training phase.
- There are **10000 images** in the test dataset.
- For each item in the test dataset, we compute the output of the model.

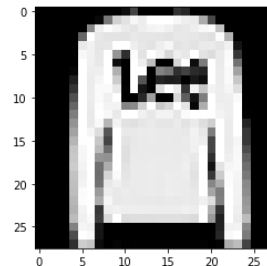
Test Data

(X_test,
Input



y_test)
True Output

2 (pullover)



Neural
Network

Predicted
Output

Test Dataset

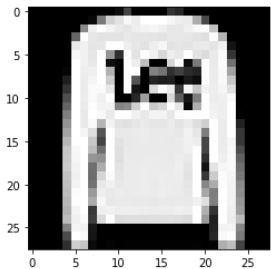
- If the **error** for the **test dataset** is large, it means that the **model** is not **predicting** well.
- We may need to **change the model** or **training data**.

Test Data

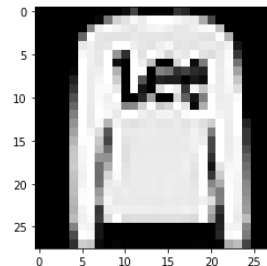
(X_{test} , y_{test})

Input

True Output



2 (pullover)



Neural
Network

Predicted
Output

Workflow

Import Data (input, output components)



Specify the model and its parameters



Fit the model to data

How to build the model?

- To build the model, we need to **specify** the **number of neurons** in each **layer**, **activation functions**, etc.

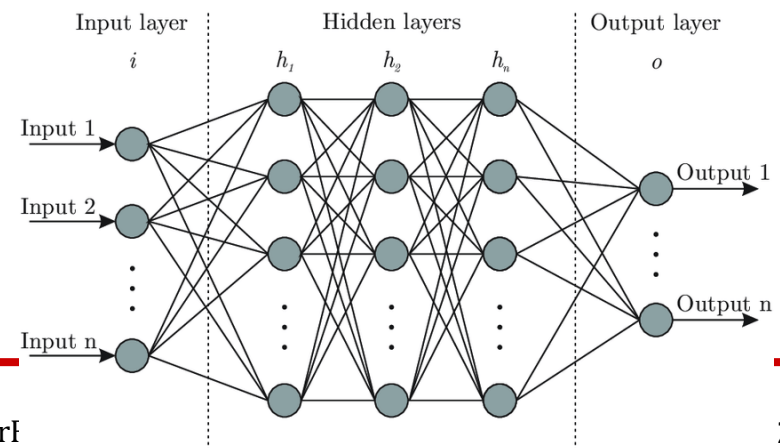
```
model = keras.models.Sequential([  
    keras.layers.Flatten(input_shape=[28, 28]),  
    keras.layers.Dense(300, activation="relu"),  
    keras.layers.Dense(100, activation="relu"),  
    keras.layers.Dense(10, activation="softmax")  
])
```

Annotations for the code:

- Sequential model**: points to the `keras.models.Sequential` constructor.
- Input layer**: points to the `Flatten` layer.
- First hidden layer**: points to the first `Dense` layer (300 neurons).
- Second hidden layer**: points to the second `Dense` layer (100 neurons).
- output layer**: points to the final `Dense` layer (10 neurons).

Annotations for the parameters:

- Number of neurons**: points to the first argument of `Dense` (300 and 100).
- Activation function**: points to the `activation` parameter ("relu" and "softmax").



Workflow

Import Data (input, output components)



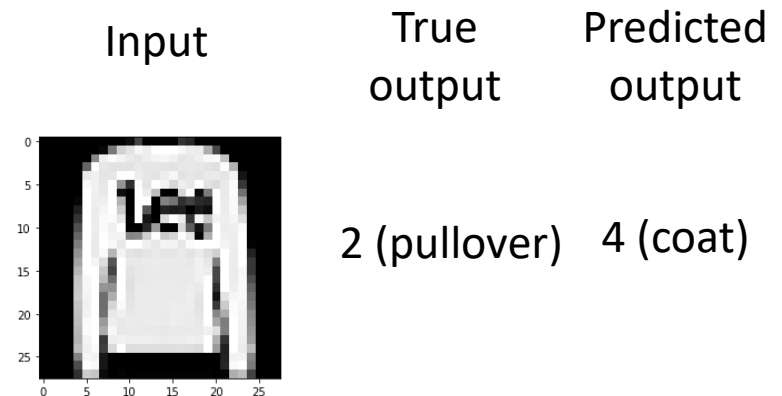
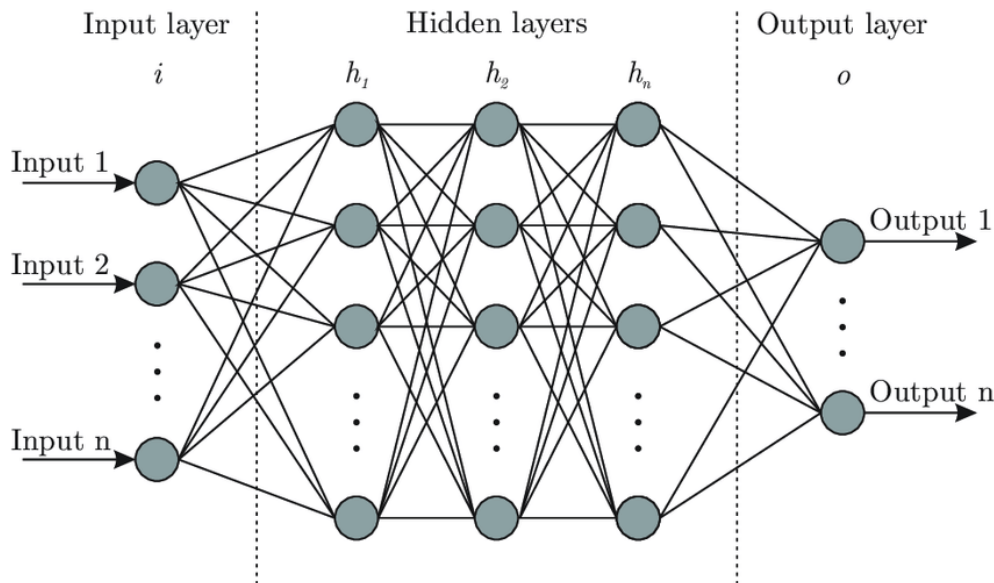
Specify the model and its parameters



Fit the model to data

How to find the weights?

- The **predicted output** might be different from the **true output**.
- We can **compute** the **error** between the **predicted output** and **true output**.
- We find the **weights** such that the **error** is **minimized**.



Example

- We next **compile** the **model** and **fit** it to the **training data**.

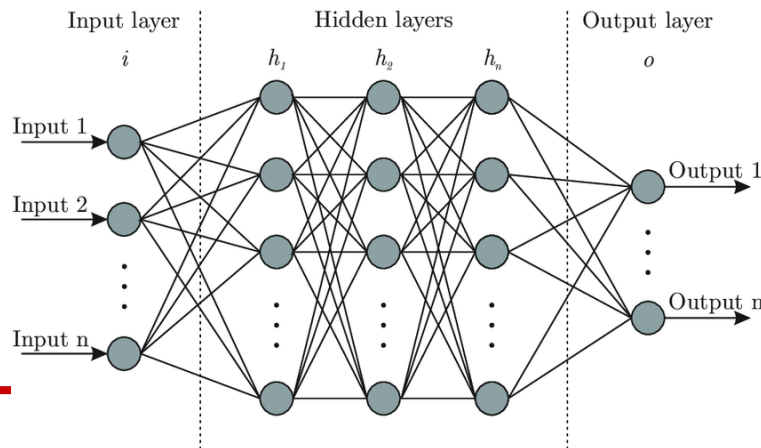
How to compute the error

```
model.compile(loss="sparse_categorical_crossentropy", optimizer="sgd",  
metrics=["accuracy"])
```

Optimization method

Stochastic Gradient Descent
(differentiation)

- `history = model.fit(X_train_full, y_train_full, epochs=30)`

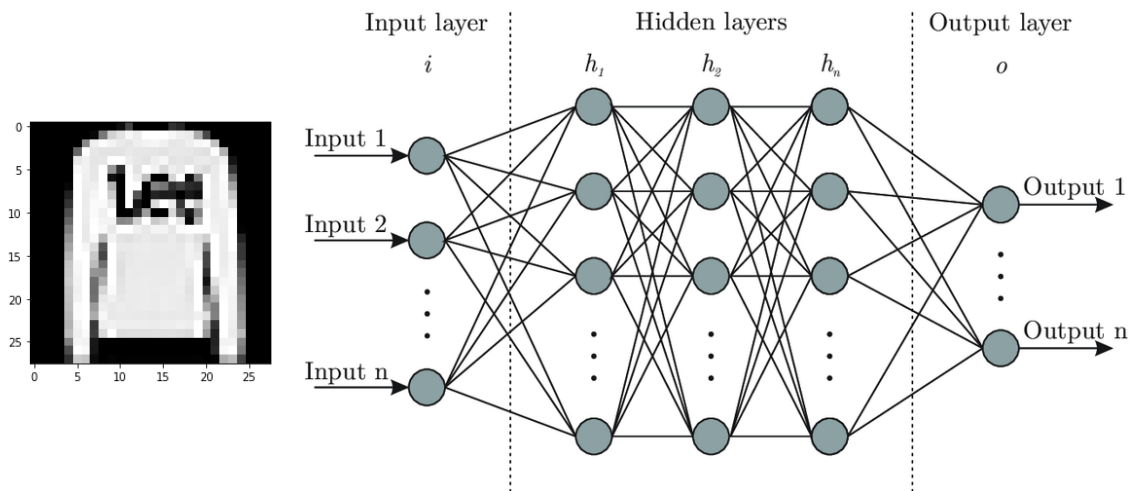


Iterations

Keras and TensorFlow

Example

- This example gives us a **probabilistic model**.
- The **model predicts** the **likelihood** that **each input belongs** to **each label**.

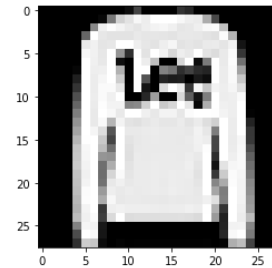


Label	Description	
0	T-shirt/top	0.05
1	Trouser	0.05
2	Pullover	0.8
3	Dress	0.05
4	Coat	0.05
5	Sandal	0
6	Shirt	0
7	Sneaker	0
8	Bag	0
9	Ankle boot	0

Example

- Cross-entropy is a popular way to quantify the error when the output is a likelihood.

- The error in this case is $-\log(0.8)$.



Label	Description	
0	T-shirt/top	0.05
1	Trouser	0.05
2	Pullover	0.8
3	Dress	0.05
4	Coat	0.05
5	Sandal	0
6	Shirt	0
7	Sneaker	0
8	Bag	0
9	Ankle boot	0

- You can read more about different ways of quantifying the error here
https://www.tensorflow.org/api_docs/python/tf/keras/losses

Example

- Stochastic gradient descent (SGD) is an optimization algorithm.
- It works based on differentiation.
- It find the weights such that the error is minimized.
- We will learn about it in the next lecture.

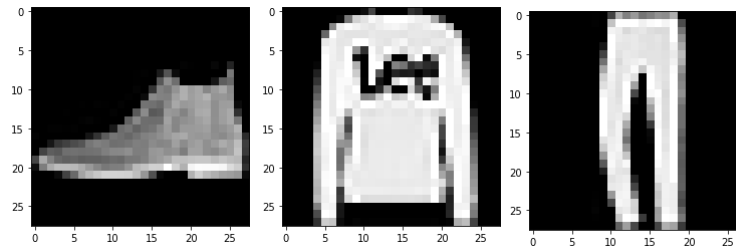
Example

- `import tensorflow as tf`
- `from tensorflow import keras`
- `from matplotlib import pyplot`
- `fashion_mnist = keras.datasets.fashion_mnist`
- `(X_train_full, y_train_full), (X_test, y_test) = fashion_mnist.load_data()`
- `X_train_full = X_train_full / 255.0`
- `model = keras.models.Sequential([`
- `keras.layers.Flatten(input_shape=[28, 28]),`
- `keras.layers.Dense(300, activation="relu"),`
- `keras.layers.Dense(100, activation="relu"),`
- `keras.layers.Dense(10, activation="softmax")`
- `])`
- `model.compile(loss="sparse_categorical_crossentropy", optimizer="sgd", metrics=["accuracy"])`
- `history = model.fit(X_train_full, y_train_full, epochs=30)`

Example

- We can use the **model** for **prediction**.
- The model predicts the **likelihood** that an **input belongs** to a **category**.

- `X_new = X_test[:3]`
- `y_proba = model.predict(X_new)`
- `y_proba.round(2)`
- `print(y_test[:3])`
- `print(y_proba)`



Label	Description
0	T-shirt/top
1	Trouser
2	Pullover
3	Dress
4	Coat
5	Sandal
6	Shirt
7	Sneaker
8	Bag
9	Ankle boot

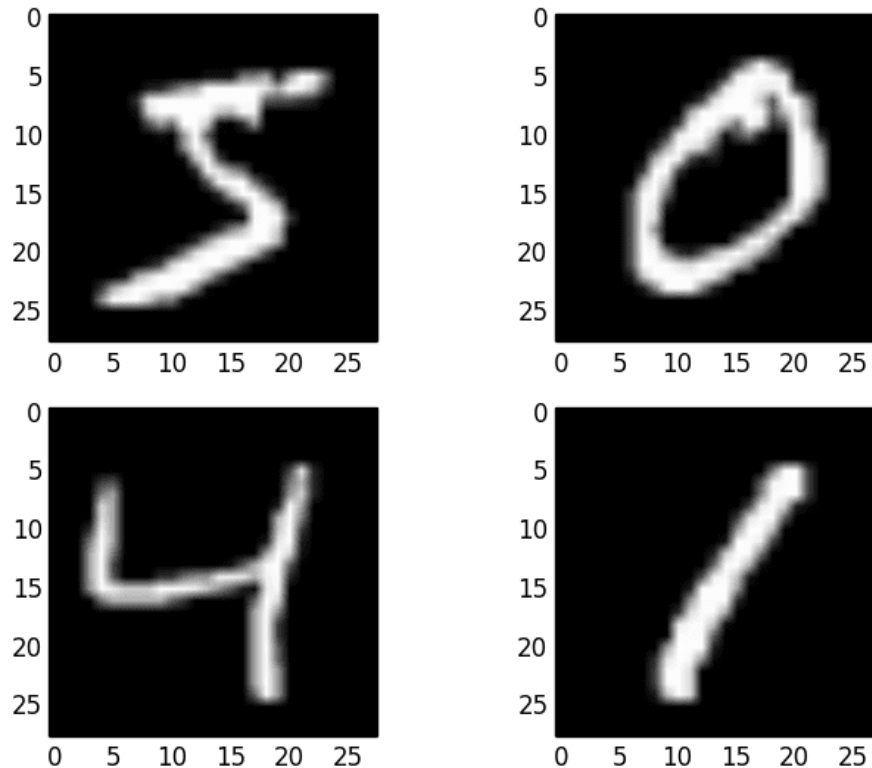
True output Predicted output

```
[9 2 1]
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]]
```

Example: Handwritten Digit Recognition

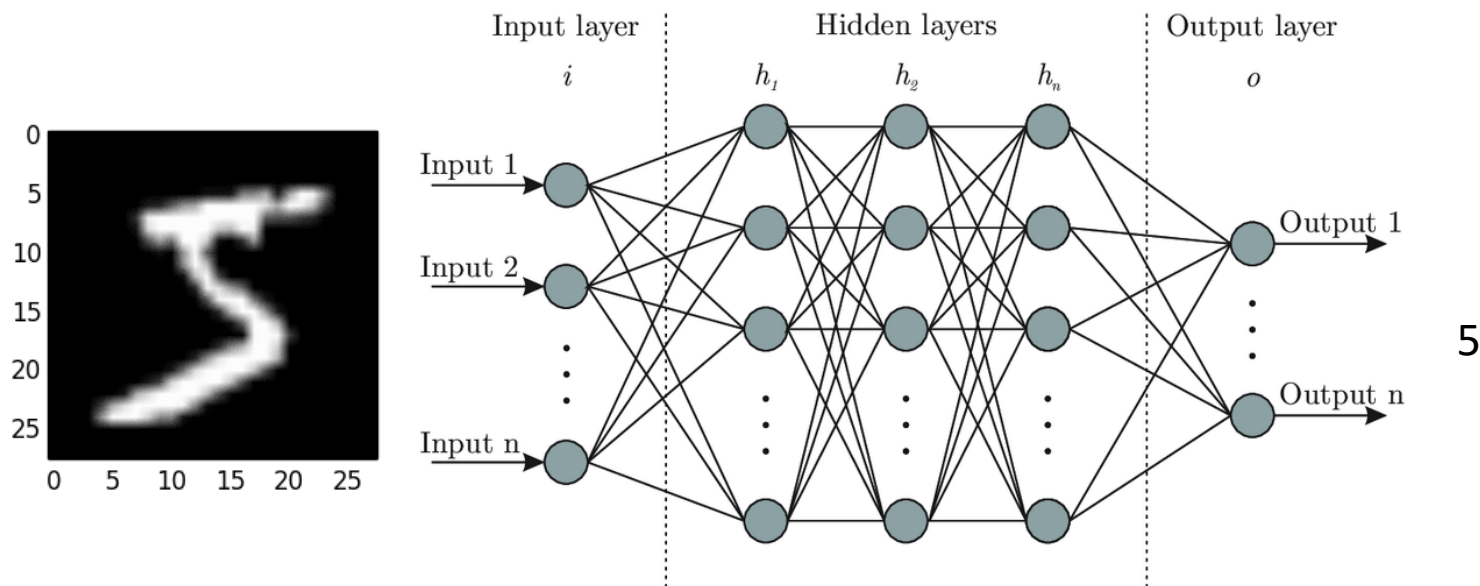
Example

- **mnist** is a dataset that contains images of handwritten digits.
- Each image in the dataset is a handwritten number from 0 to 9.



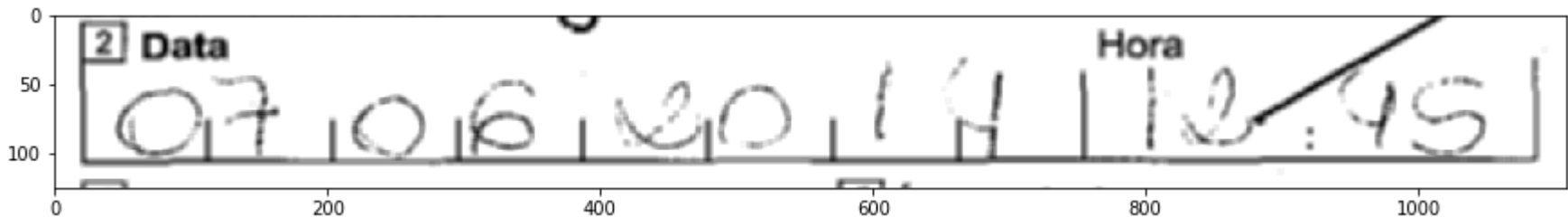
Example

- We would like to build a **model** to **predict the number** in each image in our dataset.
- This allows us to **identify** handwritten numbers.



Example

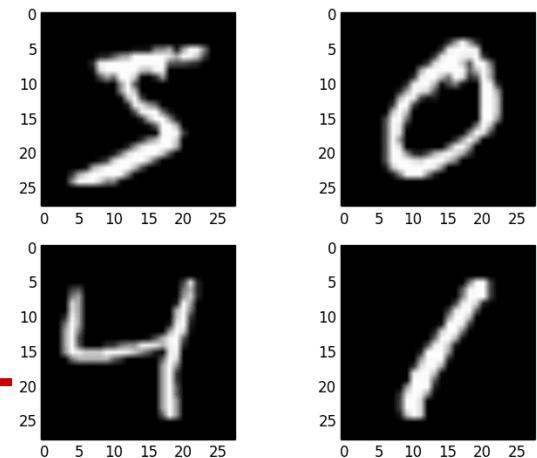
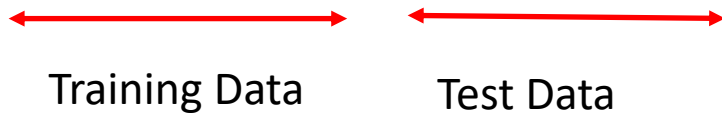
- This is very helpful for **postal services**.
- A **computer** can **identify the destination** of a **parcel** using the **image** of **handwritten postcode**.



Example

- You can use the following commands to load the **mnist** dataset and use it in your code.

```
import tensorflow as tf
from tensorflow import keras
from matplotlib import pyplot
from tensorflow.keras.datasets import mnist
(trainX, trainy), (testX, testy) = mnist.load_data()
```



Example

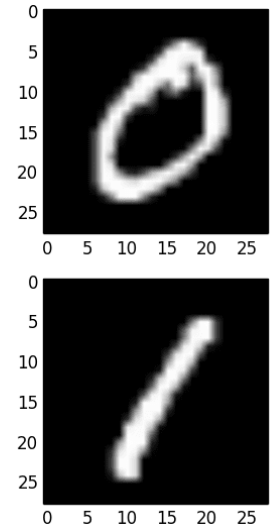
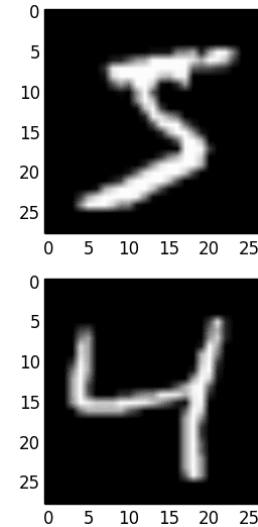
`(X_train_full , y_train_full ), (X_test , y_test ) = mnist.load_data()`



Training Data



Test Data



- `X_train_full` and `X_test` contain a set of images.
- `y_train_full` contains the label of each image in `X_train_full`.
- `y_test` contains the label of each image in `X_test`.

Example

- Let's look at some of these data

```
# plot first few images
```

```
for i in range(9):
```

```
# define subplot
```

```
    pyplot.subplot(330 + 1 + i)
```

```
# plot raw pixel data
```

```
    pyplot.imshow(trainX[i], cmap=pyplot.get_cmap('gray'))
```

```
# show the figure
```

```
pyplot.show()
```

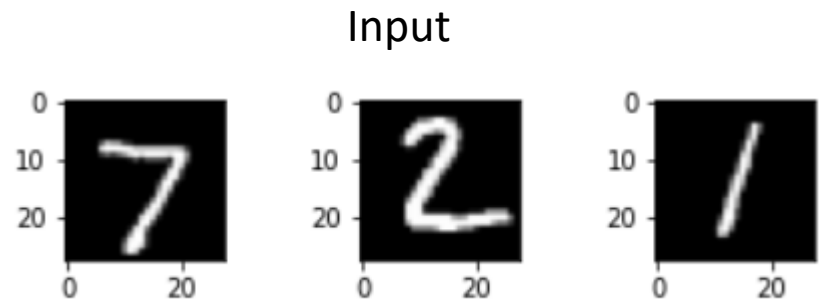
Example

- `import tensorflow as tf`
- `from tensorflow import keras`
- `from matplotlib import pyplot`
- `from tensorflow.keras.datasets import mnist`
- `(X_train_full, y_train_full), (X_test, y_test) = mnist.load_data()`
- `X_train_full = X_train_full / 255.0`
- `model = keras.models.Sequential([`
`keras.layers.Flatten(input_shape=[28, 28]),`
`keras.layers.Dense(300, activation="relu"),`
`keras.layers.Dense(100, activation="relu"),`
`keras.layers.Dense(10, activation="softmax")`
`])`
- `model.compile(loss="sparse_categorical_crossentropy", optimizer="sgd", metrics=["accuracy"])`
- `history = model.fit(X_train_full, y_train_full, epochs=30)`

Example

- We can use the **model** for **prediction**.
- The model predicts the **likelihood** that an **input image** to be a number from 0 to 9.

- `X_new = X_test[:3]`
- `y_proba = model.predict(X_new)`
- `y_proba.round(2)`
- `print(y_test[:3])`
- `print(y_proba)`



Predicted output

```
[7 2 1]
[[0. 0. 0. 0. 0. 0. 0. 1. 0. 0.]
 [0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]]
```



Resources

- Book:

“Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems” by Aurelien Geron

- It is available at CityU library.

- Canvas link:

<https://canvas.cityu.edu.hk/courses/39832/pages/tensorflow-resources>

- **Feel free** to **add more information** to the Canvas page.

- Many examples on **YouTube, GitHub, Google**, etc.

Conclusions

- Simple models may not always be helpful.
- Complex models can be built using neural networks.
- Weights are found using differentiation.
- You can build a complex neural network in Python using a few lines of code.

END

See you next week 😊!