Implementing Single-Layer and Multi-Layer Perceptron in TensorFlow

IMPORTANT

- This module is a self-paced exercise.
- Assignment #5-1 and 5-2 are included inside this module.

Module 8-4

CS 455/595a

Outline

Learning Outcomes

Upon competition of this module, students shall be able to:

Implement single-layer perceptron

Implement multi-layer perceptron

Outline

What is TensorFlow?

TensorFlow

- Open Source numerical computation library
  - Supported by Google (used in many of its products)
  - Models problems as a graph of computations and can execute models (including types of optimization) with algorithms optimized in C++
  - Primarily used in Python, but also C++ API

- Heavily evolving, contributed projects are integrated into the ecosystem with major updates
  - e.g. You can write your models using the Keras API within TensorFlow
    - Oddly, Keras as its own standalone projet uses TensorFlow and Theano as possible backends

- Used widely for artificial neural network design and implementation

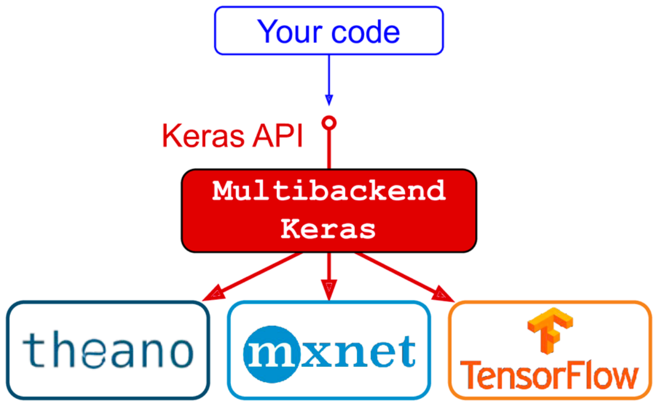
- Problems can be divided into computational chunks and distributed across CPUs and/or GPUs automatically, or distributed across multiple computers

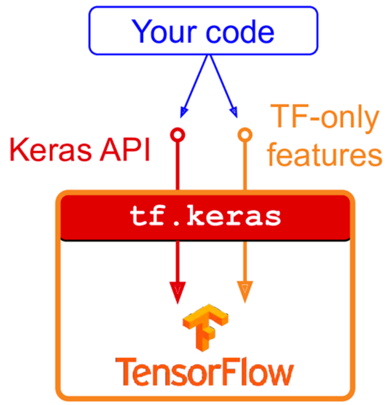
Why TensorFlow?

- Not the first or only software library and framework for ANN implementation or computation, so why is it becoming so often used?
- Scalable
  - Millions of parameters
  - Supports billions of training instances
  - Training instances can be massively large with millions of features
- Ubiquitous
  - Functions across numerous operating systems
  - Supports execution on mobile devices with full or mobile optimized networks
- Optimized with C++ so highly efficient
- Supports graph nodes for optimization such as nodes that solve for solutions to minimize a cost function
- Automatically computes partial derivatives for gradients, i.e. autodifferentiation or autodiff
- Can support operation on CPUs, GPUs, or Google's Tensor Flow Units (TFUs)
- TensorBoard supports network visualization, debugging, and learning curves

What is TensorFlow?

- A major limitation of tensorflow (in particular for version 1.x) was that it was far too complicated for many users
    - Clunky syntax
    - Numerous parts and pieces causing greater complexity
- Keras was a high-level API that provided a much lighter weight syntax for defining a variety of models
    - Implemented with two backend ANN engines, Tensorflow, Theano, and mxnet
- More recently, Tensorflow has integrated an implementation of the Tensorflow API, but with only Tensorflow operating on the backend
    - Enables seamless integration of the Keras API for defining models with additional tools/features for power users available in Tensorflow





Getting Started with TensorFlow

- The textbook has an overview of how to set up tensorflow if you are maintaining your own development environment outside of Anaconda. The textbook has several chapters on Tensorflow and its tools if you wish to go deeper.

- If you are using Anaconda, you can install tensorflow from the Anaconda power shell with the command "conda install tensorflow"

- Enabling GPU (optional)
  - If you have a graphics card with a graphics processing unit that is CUDA compatible such as the NVIDIA Geforce line of video cards, you can take advantage of the GPU to speed up your neural network.
  - Additional steps involved. Documented online
  - Alternatively, you can use Google Collaboratory. You can set your notebook to connect to one of its GPU-based or Tensorflow Processing Unit (TPU)-based virtual machines

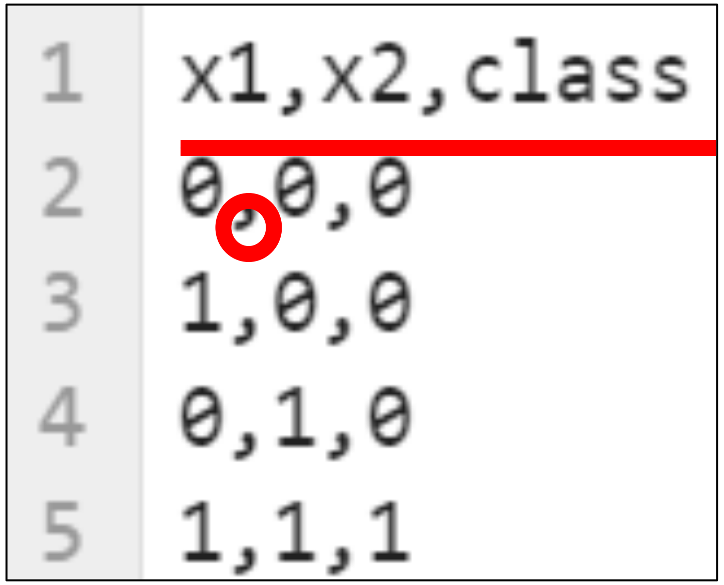
Using Keras API

- Keras is another high-level API for building our ANN models
- Advantages as cited by TensorFlow for integration of the API under its main module set:
    - User friendliness
        - API desired for simplicity
        - Implements many common ANN constructs
    - Modularity and composability
        - Flexibility to compose with user friendly API components and TF components
    - Easy to extend
        - API is extensible to create new computational units (e.g. new neuron types, new layers, new optimizers, etc.)
        - Can leverage expressivity of TensorFlow to build new Keras API compatibile modules

Outline

Implementing a single-layer perceptron E.g.) AND

1. Create a csv file ***and.csv*** consisting of the dataset below and save it.
2. From the list of files on the left of the Jupyter window, right-click ***and.csv*** under the data folder and open it with Open With → Editor.
3. Confirm that the data is divided into three columns, **separated by commas (,)**, and that the data **starts with column header**.

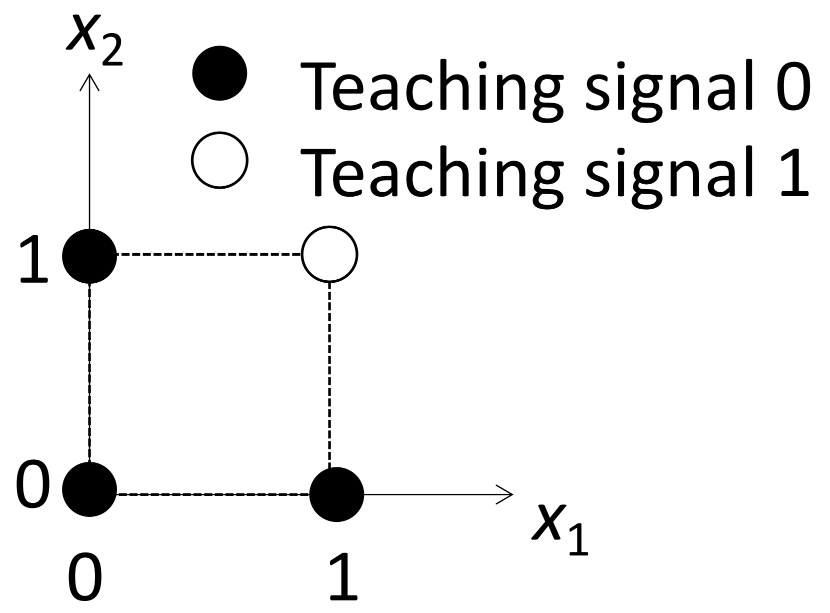


```
x1,x2,class
0,0,0
1,0,0
0,1,0
1,1,1
```

These 4 data of and.csv are the training data.

Implementing a single-layer perceptron E.g.) AND

```
x1,x2,class
0,0,0
1,0,0
0,1,0
1,1,1
```



「class」 column are the values corresponding to teaching signals

When (x_1, x_2) is (1, 1), teaching signal is 1.
When (x_1, x_2) are (0, 0), (0, 1), (1, 0), teaching signals are 0. This is AND() function.

Implementing a single-layer perceptron E.g.) AND

Create a file, ***last_name_and.ipynb***, and work on it

and.csv

```
x1,x2,class
0,0,0
1,0,0
0,1,0
1,1,1
```

Use confirmed AND() function data to train a neural network.

- "Markdown" cell：

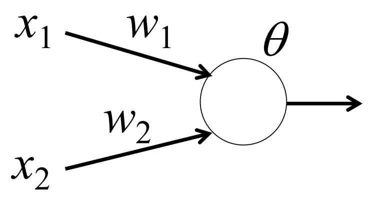
```
import modules
```

- "Code" cell：

```
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
```

Import two classes from tensorflow.keras.models and tensorflow.keras.layers, respectively.

Implementing a single-layer perceptron E.g.) AND



- "Markdown" cell: Load and split data, then check the number of data
- "Code" cell：

```
dat=np.loadtxt('data/points_and.csv', delimiter=',', skiprows=1)
data_train=dat[:,:-1]            Extract all columns except the last one (2d)
class_train=dat[:,-1]            Extract the last one column only (1d)
print('data:', data_train.shape) → (4, 2)
print('class:', class_train.shape) → (4, )
```

Columns are separated with ',', so set delimiter=','.
The first line is labels of columns so use skiprows=1 to skip it.
Load data into variable dat, and create 2 arrays via array slicing, a 2-column data(named data_train) and a 1-column data(class_train).
If data_train is 4x2 2D array and class_train is 1D array with 4 elements, data are split correctly.

- "Markdown" cell： Model building and learning

Review) Array slicing

```
          Column  1       2         3
           dat = [ data1  data2  class ]
                       |      |      |      |
Index                  0      1      2      3
                      -3     -2     -1
```

Specify a range of indices (return a 2D array)

dat[: , **:-1**] → [data1 data2]

dat[: , **-1**] → [class]

Specify a column index
(return an 1D array)

Implementing a single-layer perceptron E.g.) AND

- "Code" cell：

```
model=Sequential()
model.add(Dense(1, input_shape=(2,), activation='sigmoid'))
model.compile(optimizer='sgd', loss='binary_crossentropy',
          metrics=['accuracy'])
fit_log=model.fit(data_train, class_train, epochs=1000, batch_size=1)
```

Records(logging) of learning process

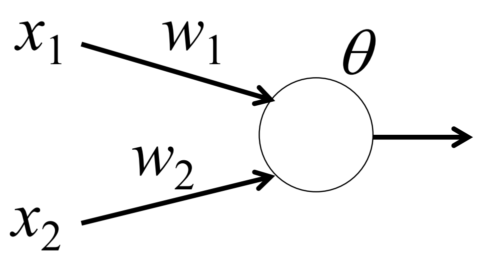
First, create a Sequential object named model via constructor.
Define layers of neurons via model.add().
Dense() function define neurons.

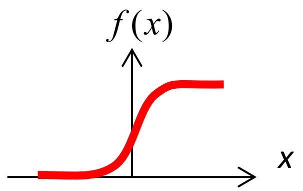
Dense(1, input_shape=(2,), activation='sigmoid')

One neuron in this layer

Input is 2D

Activation function is sigmoid function.





Implementing a single-layer perceptron E.g.) AND

Optimize weights(w) and threshold(θ) with the method sgd
Error function (teaching signals and outputs) is binary_crossentropy.
Accuracy is displayed during learning.

Optimization in Keras
https://www.tensorflow.org/api_docs/python/tf/keras/optimizers

Error function or loss function in Keras
https://www.tensorflow.org/api_docs/python/tf/keras/losses

```
model.fit(data_train, class_train,epochs=1000, batch_size=1)
```

Takes data_train as a training data, and class_train as teaching signals to train the model. Epochs is the number of update for weights and thresholds. batch_size is the number of data to learn at each time (Set it to 1 is OK since we have only 4 data samples）

Implementing a single-layer perceptron E.g.) AND

```
…
Epoch 1000/1000 4/4 [======] - 0s - loss: 0.2693 - acc: 1.0000
```

The value from the error function （smaller is better. ）

Accuracy （If you don’t get 1.00, learning mistakes occurred. ）

Re-execute the cell by press "Shift+Enter" if acc is not 1.0.

Implementing a single-layer perceptron E.g.) AND

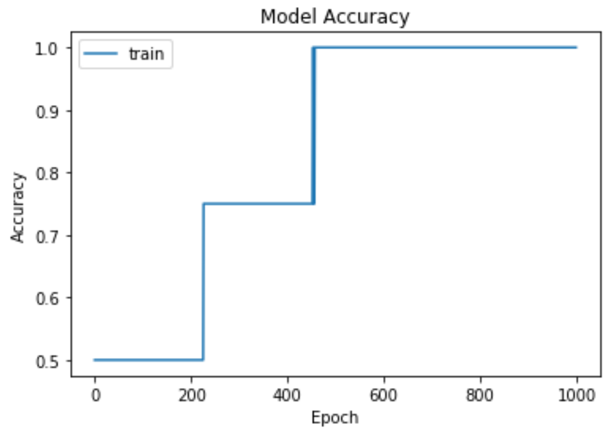
- "Markdown" cell：

```
Graph for accuracy
```

- "Code" cell：

```
plt.plot(fit_log.history['accuracy'], label='train')
plt.title('Model Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

Among records of learning process, read accuracy(accuracy) values and plot it against epoch.



If the prediction accuracy reaches 1.0 at the end, then the training is complete

The graph is plotted with epoch(# of repetitive learning) against accuracy
(The shape of the graph will be changed during executions)

Implementing a single-layer perceptron E.g.) AND

- "Markdown" cell： Find the predicted value for the training data
- "Code" cell：

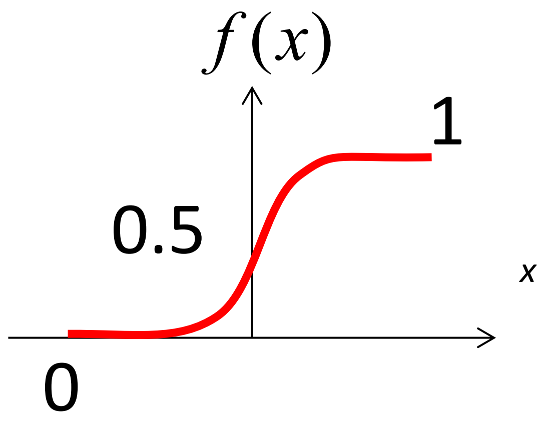
```
pred=model.predict(data_train)
print(pred)
```

Regarding each data in data_train, compute the predicted values (decision values) and save to the variable pred

```
[[ 0.06211448]
 [ 0.24230061]
 [ 0.25451156]
 [ 0.62242359]]
```

Activation function is sigmoid function.

1 if ≥0.5,
0 otherwise.



*** The values change with each run. But they should be essentially the same as teacher signals (0, 0, 0, 1). Let's confirm this.**

(Optional) Draw a Separating Hyperplane

- "Markdown" cell：

```
Prepare to draw a separating
hyperplane: teaching signal
```

- "Code" cell：

```
t1=data_train[class_train_1d==1]
t0=data_train[class_train_1d==0]
t1_x=t1[:,0]   ← X-coordinate of a point(teaching signal of 1)
t1_y=t1[:,1]   ← Y-coordinate of a point(teaching signal of 1)
t0_x=t0[:,0]   ← X-coordinate of a point(teaching signal of 0)
t0_y=t0[:,1]   ← y-coordinate of a point(teaching signal of 0)
```

1, 2 lines: extract data from data_train（training data） corresponding to teaching signals（class_train_1d） of 1 or 0.

(Optional) Draw a Separating Hyperplane

How to create a partial array that meets conditions

x0 = **x**[**cls**==0]

Class numbers of each point stored in cls

cls: [..., **0**, 1, 1, 1, **0**, 1, **0**, ...]

x: [..., **5.0**, 7.0, 6.4, 6.9, **5.5**, 6.5, **5.7**, ...]

x coordinates are stored in array x.

x0: [..., **5.0**, **5.5**, **5.7**, ...]

From array **x,** Extract x-coordinates from array x by indices, which are responding to indices of elements which cluster number is 0 in **cls** and save them to Array **x0**

(Optional) Draw a Separating Hyperplane

- "Markdown" cell：

```
Prepare to draw a separating hyperplane: grid points
```

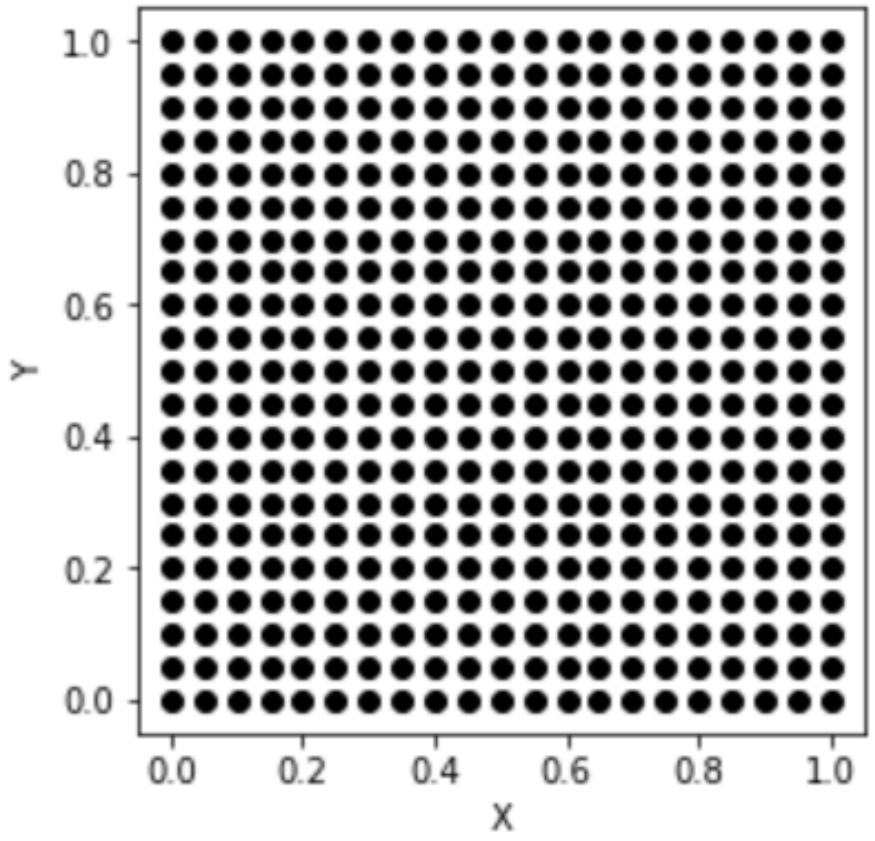
- "Code" cell：

```
g=np.loadtxt('data/grid01_21x21.csv', delimiter=',')
pred_g=model.predict(g)[:, 0]
g1=g[pred_g>=0.5]     ← Predicted value(>=0.5) belongs to 1
g0=g[pred_g<0.5]      ← Predicted value(<0.5) belongs to 0
g1_x=g1[:, 0]   ← X-coordinate of predicted points belongs to 1
g1_y=g1[:, 1]   ← Y-coordinate of predicted points belongs to 1
g0_x=g0[:, 0]   ← X-coordinate of predicted points belongs to 0
g0_y=g0[:, 1]   ← Y-coordinate of predicted points belongs to 0
```

- In grid01_21x21.csv, create 21 grid points along X, Y direction between 0 and 1.
- Lines 1 and 2: Load these grid points to g, use model.predict(g) to obtain the prediction values and save them to array pred_g, transform them to 1D arrays.

(Optional) Draw a Separating Hyperplane

Grid data points
grid01_21x21.csv



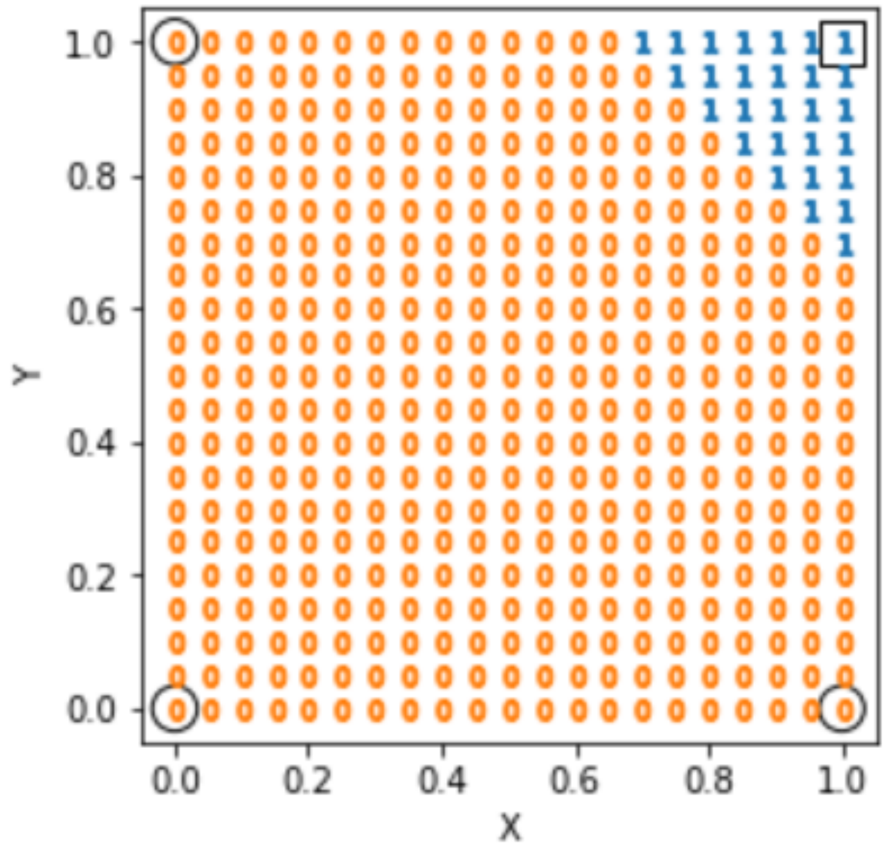
(Optional) Draw a Separating Hyperplane

- "Markdown" cell : draw a separating hyperplane
- "Code" cell ：

```
plt.scatter(t1_x,t1_y,marker='s',facecolors='none',edgecolors='black',s=180)
plt.scatter(t0_x,t0_y,marker='o',facecolors='none',edgecolors='black',s=180)
plt.scatter(g1_x,g1_y,marker='$1$')
plt.scatter(g0_x,g0_y,marker='$0$')
plt.gca().set_aspect('equal', adjustable='box')
plt.xlim(-0.05,1.05)
plt.ylim(-0.05,1.05)
plt.xlabel('X')
plt.ylabel('Y')
plt.show()
```

- Set points of teaching signal of 1 to size180, a unfilled black square
- Set points of teaching signal of 0 to size 180, a unfilled circle.
- Set grid points of 1 to 「1」
- Set grid points of 0 to 「0」
- Set graph aspect ratio to 1:1（square）.

(Optional) Draw a Separating Hyperplane



Consistent with teaching signals （□:1, ○:0）

The separating hyperplane is a straight line.

Assignment #5-1

Complete the learning (training) of a single-layer perceptron for the training data (AND function), and submit the output values as a part of ***last_name_and.ipynb***

Find predicted values for training data

```
pred=model.predict(data_train)
print(pred)
```

```
[[ 0.06211448]
 [ 0.24230061]
 [ 0.25451156]
 [ 0.62242359]]
```

Submit these values

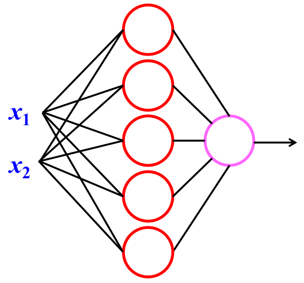
The values change with execution.

E.q.) Your file submission includes the following "Markdown" cell:

```
[[ 0.06211448]
 [ 0.24230061]
 [ 0.25451156]
 [ 0.62242359]]
```

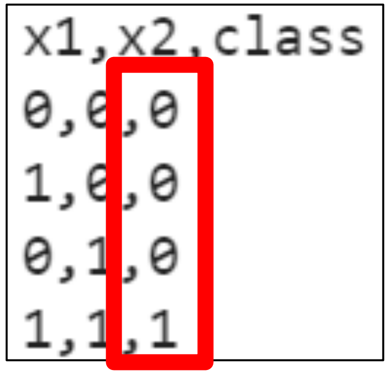
Outline

Implementing a multi-layer perceptrons E.g.) XOR



Create a XOR data file

1. Right-click and.csv under the data folder and select Duplicate.
2. Right-click the generated ***and-Copy1.csv***, and rename it to ***xor.csv***.
3. Right-click xor.csv and open with Open With> Editor.



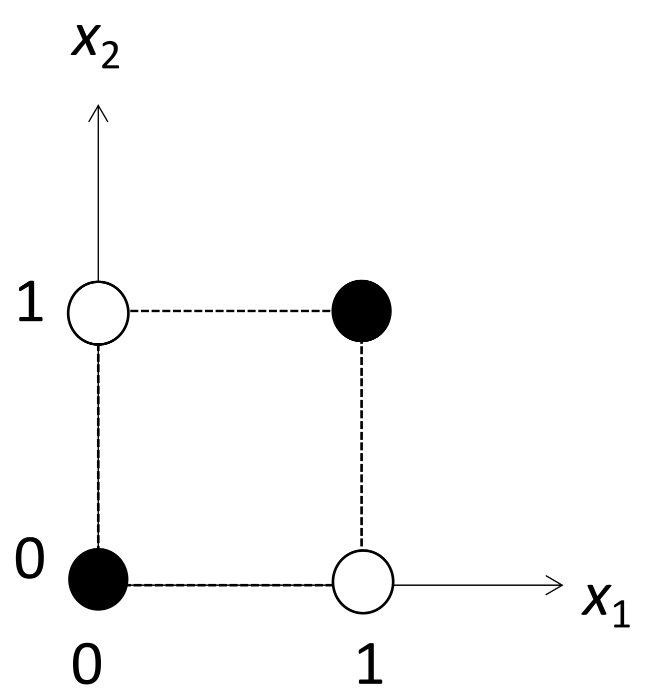
```
x1,x2,class
0,0,0
1,0,0
0,1,0
1,1,1
```

Change 0, 0, 0, 1 to **0, 1, 1, 0** in the red frame、Save it by select (menu) File --> Save CSV File. This completes the creation of the XOR function data file.

Implementing a multi-layer perceptrons E.g.) XOR

data/xor.csv

```
x1,x2,class
0,0,0
1,0,1
0,1,1
1,1,0
```



● Teaching signal 0

○ Teaching signal 1

The values in "class" column are corresponding to teaching signals

When (x_1, x_2) are (1, 0), (0, 1), teaching signals are 1.
When (x_1, x_2) are (0, 0), (1, 1),teaching signals are 0.
This is XOR() function.

Implementing a multi-layer perceptrons E.g.) XOR

Create a file, ***last_name_xor.ipynb***, and work on it

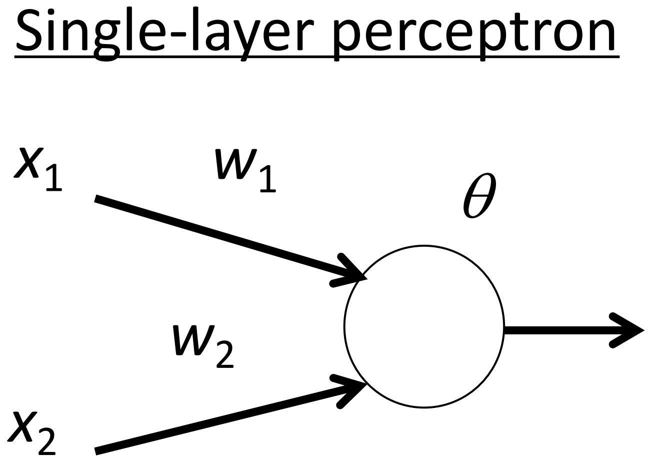
Add a middle layer to the single-layer perceptron

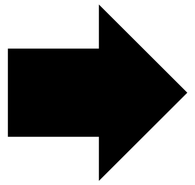
- Add a middle layer to the single-layer perceptron to create a multilayer perceptron, try to learn with XOR() function.
- Edit codes according to the following codes.
- "Markdown" cell: Model building and learning
- "Code" cell:

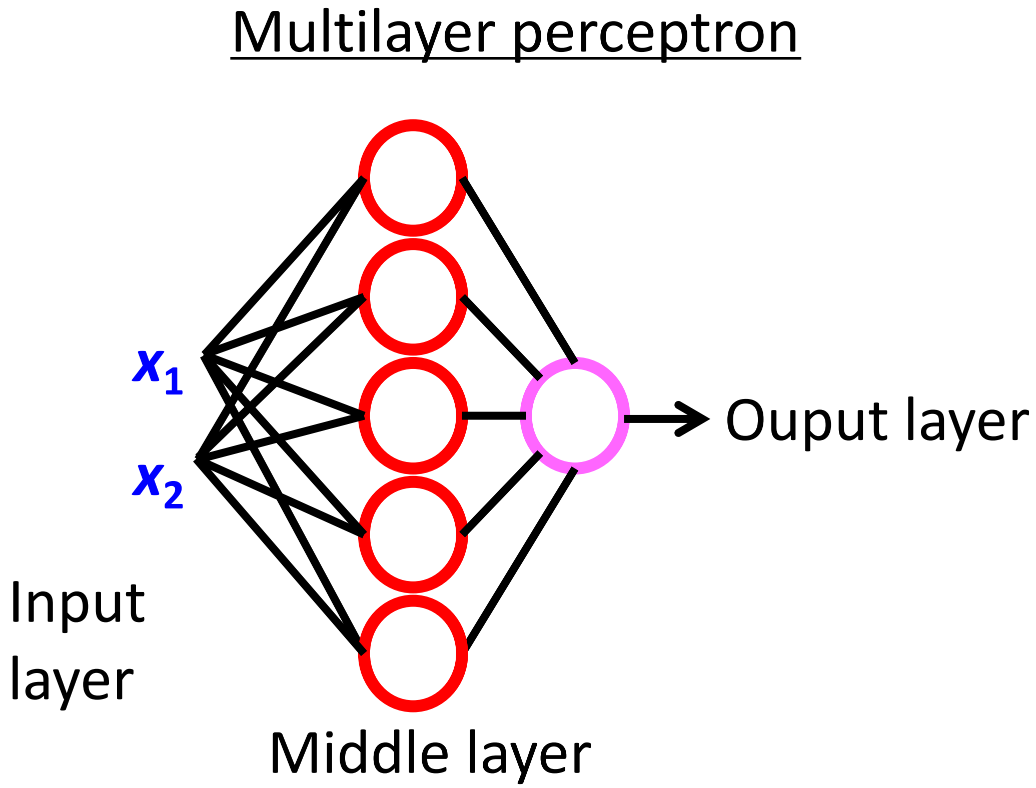
```
model=Sequential()
model.add(Dense(5, input_shape=(2,), activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='rmsprop', loss='binary_crossentropy',
          metrics=['accuracy'])
print(model.summary())
model.fit(data_train, class_train, epochs=3000, batch_size=1)
```

Implementing a multi-layer perceptrons E.g.) XOR

```
model.add(Dense(5, input_shape=(2,), activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```







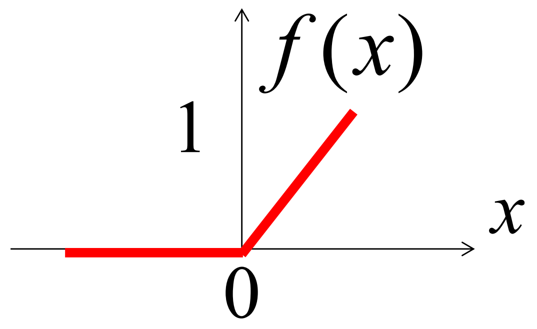
* Neurons are fully connected among layers（Fully connected, Dense）

Implementing a multi-layer perceptrons E.g.) XOR

(Changes)

Set 5 neurons at the middle layer(ReLU as activation function), rmsprop as optimizer, epochs to 3000

ReLU



* input_shape can be omitted for layers except input layer (be automatically determined if the dimension of previous layer were known.

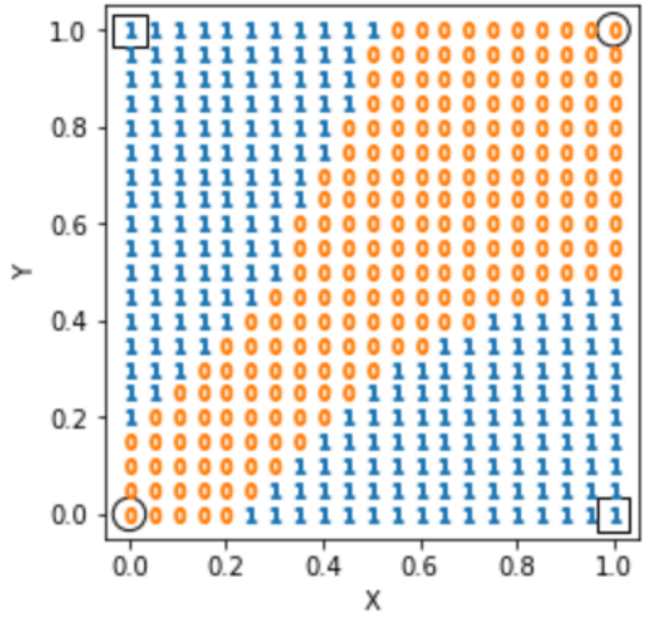
Although it is not necessary to change activation function and optimization method, we can expect the changes might improve the accuracy more or less; Try to use different parameters yourself.

Implementing a multi-layer perceptrons E.g.) XOR

- The learning (training) may be failed even though we train the classifier 3000 times with model.fit(). If so, retrain the classifier with model.fit() again.
- If it is all right, you can get the figure as shown below that the nonlinear separating hyperplane w.r.t. predicted values can be displayed, where it shows that XOR function data can be learned.

```
[[  1.64207742e-02]
 [  9.99999523e-01]
 [  9.99999762e-01]
 [  4.72090136e-07]]
```

* The values change with each execution.
From top to bottom, **<0.5, >=0.5,**
>=0.5, <0.5 if learning is completed



Assignment #5-2

Complete the learning (training) of a multilayer perceptron for the training data (XOR function), and submit the output values as a part of **last_name_xor.ipynb**

Find predicted values for training data

```
pred=model.predict(data_train)
print(pred)
```

```
[[ 7.62540076e-05]
 [ 9.99978065e-01]
 [ 9.99991655e-01]
 [ 1.04920127e-05]]
```

Submit these values
The values change with execution.

E.q.) Your file submission includes the following "Markdown" cell:

```
[[ 7.62540076e-05]
 [ 9.99978065e-01]
 [ 9.99991655e-01]
 [ 1.04920127e-05]]
```

Outline

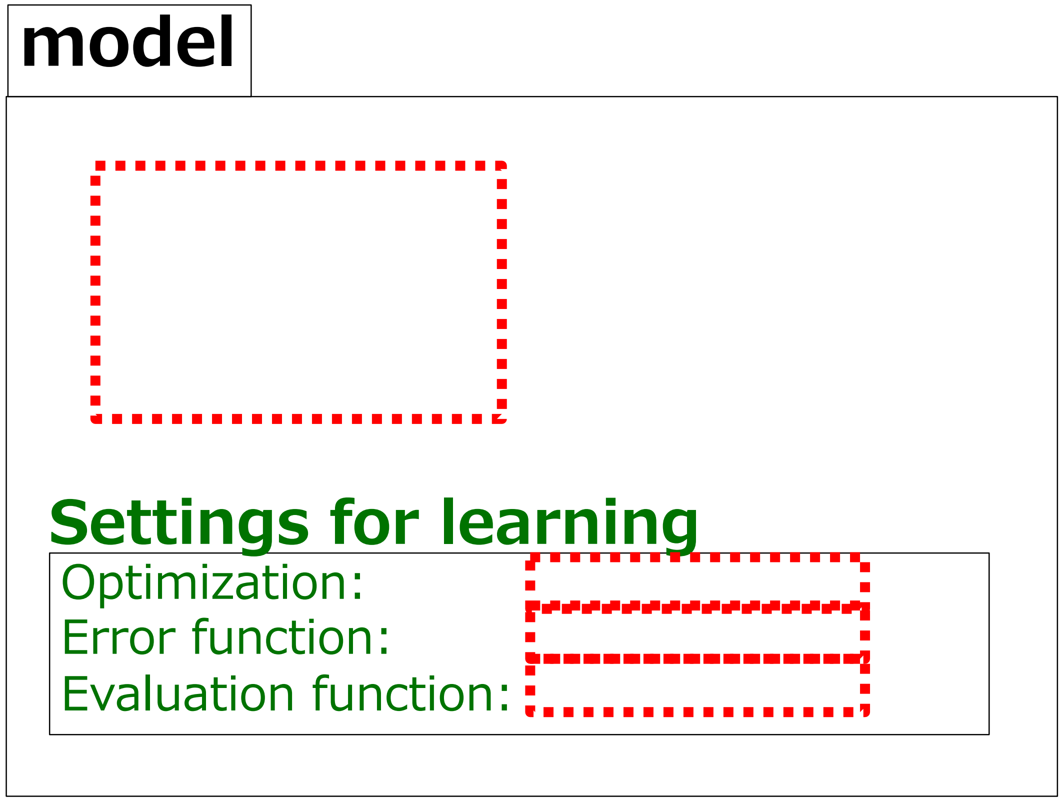
Summary

- Programming of neural network
  - MODEL = Sequential() ←Construct an object for neural network
  - MODEL.add(Dense(...)) ←Add a layer of neurons to the object
  - MODEL.compile(...) ←Settings of learning (training)
  - Variable_for_log = MODEL.fit(Training_data, Teacher_signal) ←Execution of training
  - Variable_for_results = MODEL.predict() ←Prediction for training data and output of the results

Summary of the workflow

model = Sequential()

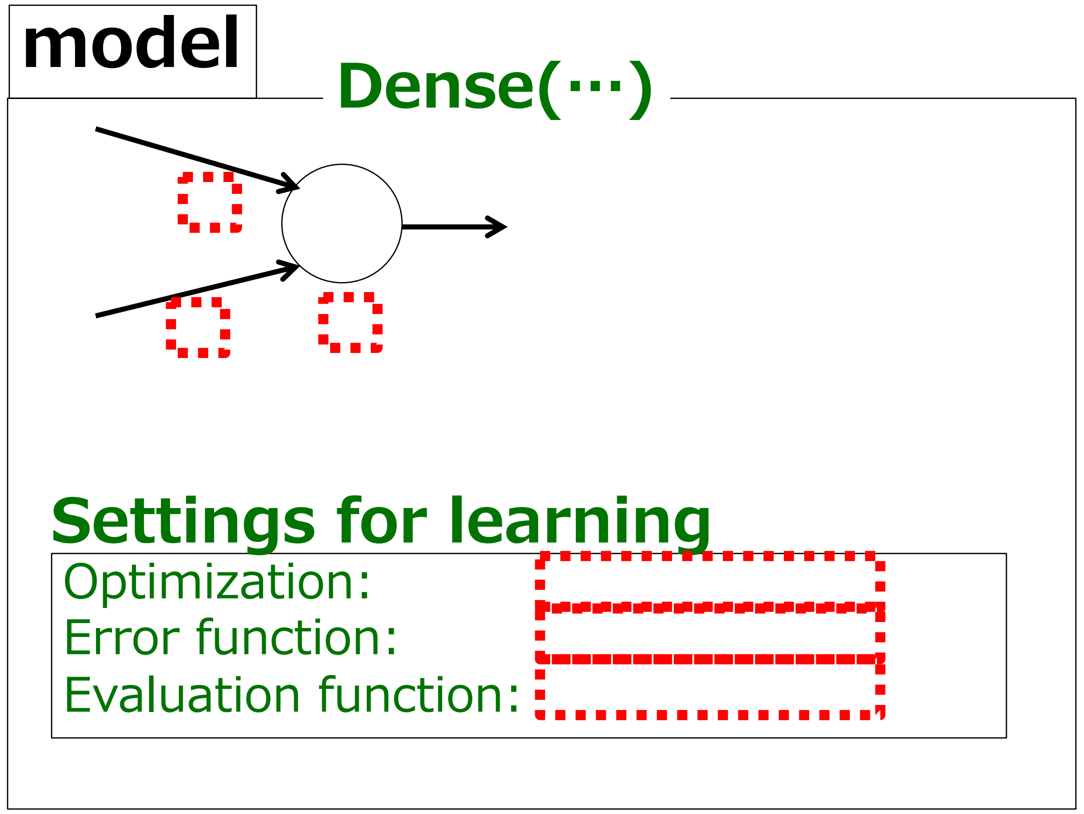
Construct a computational model of neural network



model.add(**Dense(…)**)

model = Sequential()

Add construction objects of neural network

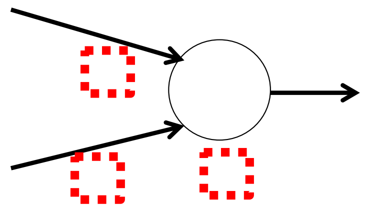


model.compile(**...**)

model = Sequential()

Set learning parameters

model



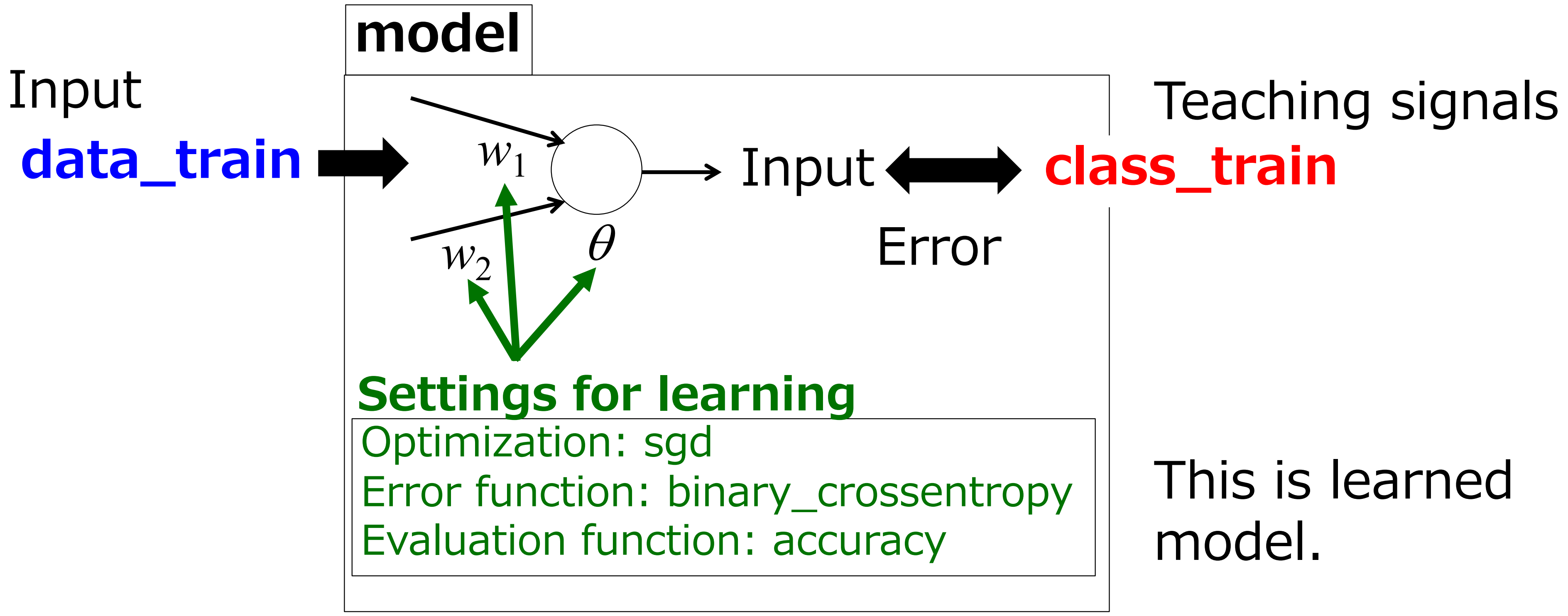
Settings for learning

Optimization: **sgd**
Error function: **binary_crossentropy**
Evaluation function : **accuracy**

optimizer: Optimization method
loss: Error function
metrics: Evaluation function

fit_log = **model**.fit(**data_train**, **class_train**, …)

Carry out learning (Assign intermediate results to variable fit_log)
＝Determine weight *w* and threshold θ via optimization to make "Error" smaller.



Optimization: sgd
Error function: binary_crossentropy
Evaluation function: accuracy