Basics of MLP

- **Objective: Create vanilla neural networks (i.e., Multilayer perceptrons) for simple regression/classification tasks with Keras**

In [2]:

```
!pip install tensorflow
```

```
Collecting tensorflow
  Downloading tensorflow-2.3.1-cp38-cp38-manylinux2010_x86_64.whl (320.5 MB)
     |████████████████████████████████| 320.5 MB 4.9 kB/s  eta 0:00:01     |██
| 21.0 MB 16.3 MB/s eta 0:00:19     |████████████████████▉        | 247.6 MB 40.9 MB/s et
a 0:00:02
Collecting absl-py>=0.7.0
  Downloading absl_py-0.10.0-py3-none-any.whl (127 kB)
     |████████████████████████████████| 127 kB 67.1 MB/s eta 0:00:01
Collecting grpcio>=1.8.6
  Downloading grpcio-1.32.0-cp38-cp38-manylinux2014_x86_64.whl (3.8 MB)
     |████████████████████████████████| 3.8 MB 7.7 kB/s  eta 0:00:01
Collecting protobuf>=3.9.2
  Downloading protobuf-3.13.0-cp38-cp38-manylinux1_x86_64.whl (1.3 MB)
     |████████████████████████████████| 1.3 MB 21.7 MB/s eta 0:00:01
Collecting opt-einsum>=2.3.2
  Downloading opt_einsum-3.3.0-py3-none-any.whl (65 kB)
     |████████████████████████████████| 65 kB 619 kB/s  eta 0:00:01
Collecting termcolor>=1.1.0
  Using cached termcolor-1.1.0.tar.gz (3.9 kB)
Requirement already satisfied: six>=1.12.0 in /home/ramesh/anaconda3/lib/python3.8/site-pack
ages (from tensorflow) (1.15.0)
Collecting gast==0.3.3
  Using cached gast-0.3.3-py2.py3-none-any.whl (9.7 kB)
Collecting astunparse==1.6.3
  Downloading astunparse-1.6.3-py2.py3-none-any.whl (12 kB)
Requirement already satisfied: h5py<2.11.0,>=2.10.0 in /home/ramesh/anaconda3/lib/python3.8/
site-packages (from tensorflow) (2.10.0)
Collecting tensorboard<3,>=2.3.0
  Downloading tensorboard-2.3.0-py3-none-any.whl (6.8 MB)
     |████████████████████████████████| 6.8 MB 38.4 MB/s eta 0:00:01
Collecting tensorflow-estimator<2.4.0,>=2.3.0
  Downloading tensorflow_estimator-2.3.0-py2.py3-none-any.whl (459 kB)
     |████████████████████████████████| 459 kB 35.6 MB/s eta 0:00:01
Collecting keras-preprocessing<1.2,>=1.1.1
  Downloading Keras_Preprocessing-1.1.2-py2.py3-none-any.whl (42 kB)
     |████████████████████████████████| 42 kB 215 kB/s  eta 0:00:01
Requirement already satisfied: numpy<1.19.0,>=1.16.0 in /home/ramesh/anaconda3/lib/python3.8
/site-packages (from tensorflow) (1.18.5)
Requirement already satisfied: wrapt>=1.11.1 in /home/ramesh/anaconda3/lib/python3.8/site-pa
ckages (from tensorflow) (1.11.2)
Collecting google-pasta>=0.1.8
  Downloading google_pasta-0.2.0-py3-none-any.whl (57 kB)
     |████████████████████████████████| 57 kB 1.3 MB/s  eta 0:00:01
Requirement already satisfied: wheel>=0.26 in /home/ramesh/anaconda3/lib/python3.8/site-pack
ages (from tensorflow) (0.34.2)
Requirement already satisfied: setuptools in /home/ramesh/anaconda3/lib/python3.8/site-packa
ges (from protobuf>=3.9.2->tensorflow) (49.2.0.post20200714)
Collecting google-auth<2,>=1.6.3
  Downloading google_auth-1.22.1-py2.py3-none-any.whl (114 kB)
     |████████████████████████████████| 114 kB 32.3 MB/s eta 0:00:01
Collecting tensorboard-plugin-wit>=1.6.0
  Downloading tensorboard_plugin_wit-1.7.0-py3-none-any.whl (779 kB)
     |████████████████████████████████| 779 kB 28.7 MB/s eta 0:00:01
```

```
Requirement already satisfied: requests<3,>=2.21.0 in /home/ramesh/anaconda3/lib/python3.8/site-packages (from tensorboard<3,>=2.3.0->tensorflow) (2.24.0)
Requirement already satisfied: werkzeug>=0.11.15 in /home/ramesh/anaconda3/lib/python3.8/site-packages (from tensorboard<3,>=2.3.0->tensorflow) (1.0.1)
Collecting markdown>=2.6.8
  Downloading Markdown-3.3.2-py3-none-any.whl (95 kB)
     |████████████████████████████████| 95 kB 876 kB/s  eta 0:00:01
Collecting google-auth-oauthlib<0.5,>=0.4.1
  Using cached google_auth_oauthlib-0.4.1-py2.py3-none-any.whl (18 kB)
Collecting rsa<5,>=3.1.4; python_version >= "3.5"
  Downloading rsa-4.6-py3-none-any.whl (47 kB)
     |████████████████████████████████| 47 kB 1.1 MB/s  eta 0:00:01
Collecting cachetools<5.0,>=2.0.0
  Downloading cachetools-4.1.1-py3-none-any.whl (10 kB)
Collecting pyasn1-modules>=0.2.1
  Downloading pyasn1_modules-0.2.8-py2.py3-none-any.whl (155 kB)
     |████████████████████████████████| 155 kB 32.6 MB/s eta 0:00:01
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /home/ramesh/anaconda3/lib/python3.8/site-packages (from requests<3,>=2.21.0->tensorboard<3,>=2.3.0->tensorflow) (1.25.9)
Requirement already satisfied: chardet<4,>=3.0.2 in /home/ramesh/anaconda3/lib/python3.8/site-packages (from requests<3,>=2.21.0->tensorboard<3,>=2.3.0->tensorflow) (3.0.4)
Requirement already satisfied: certifi>=2017.4.17 in /home/ramesh/anaconda3/lib/python3.8/site-packages (from requests<3,>=2.21.0->tensorboard<3,>=2.3.0->tensorflow) (2020.6.20)
Requirement already satisfied: idna<3,>=2.5 in /home/ramesh/anaconda3/lib/python3.8/site-packages (from requests<3,>=2.21.0->tensorboard<3,>=2.3.0->tensorflow) (2.10)
Collecting requests-oauthlib>=0.7.0
  Using cached requests_oauthlib-1.3.0-py2.py3-none-any.whl (23 kB)
Collecting pyasn1>=0.1.3
  Using cached pyasn1-0.4.8-py2.py3-none-any.whl (77 kB)
Collecting oauthlib>=3.0.0
  Using cached oauthlib-3.1.0-py2.py3-none-any.whl (147 kB)
Building wheels for collected packages: termcolor
  Building wheel for termcolor (setup.py) ... done
  Created wheel for termcolor: filename=termcolor-1.1.0-py3-none-any.whl size=4830 sha256=995566bcb20664325aed26c85d5cd80d99842f72447b8bdfeb7fc6fb0a56caf8
  Stored in directory: /home/ramesh/.cache/pip/wheels/a0/16/9c/5473df82468f958445479c59e784896fa24f4a5fc024b0f501
Successfully built termcolor
Installing collected packages: absl-py, grpcio, protobuf, opt-einsum, termcolor, gast, astunparse, pyasn1, rsa, cachetools, pyasn1-modules, google-auth, tensorboard-plugin-wit, markdown, oauthlib, requests-oauthlib, google-auth-oauthlib, tensorboard, tensorflow-estimator, keras-preprocessing, google-pasta, tensorflow
Successfully installed absl-py-0.10.0 astunparse-1.6.3 cachetools-4.1.1 gast-0.3.3 google-auth-1.22.1 google-auth-oauthlib-0.4.1 google-pasta-0.2.0 grpcio-1.32.0 keras-preprocessing-1.1.2 markdown-3.3.2 oauthlib-3.1.0 opt-einsum-3.3.0 protobuf-3.13.0 pyasn1-0.4.8 pyasn1-modules-0.2.8 requests-oauthlib-1.3.0 rsa-4.6 tensorboard-2.3.0 tensorboard-plugin-wit-1.7.0 tensorflow-2.3.1 tensorflow-estimator-2.3.0 termcolor-1.1.0
```

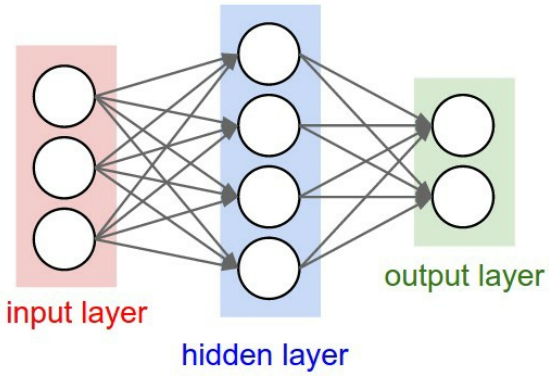
In [3]:

```
!pip install keras
```

```
Collecting keras
  Downloading Keras-2.4.3-py2.py3-none-any.whl (36 kB)
Requirement already satisfied: scipy>=0.14 in /home/ramesh/anaconda3/lib/python3.8/site-packages (from keras) (1.5.0)
Requirement already satisfied: h5py in /home/ramesh/anaconda3/lib/python3.8/site-packages (from keras) (2.10.0)
Requirement already satisfied: pyyaml in /home/ramesh/anaconda3/lib/python3.8/site-packages (from keras) (5.3.1)
Requirement already satisfied: numpy>=1.9.1 in /home/ramesh/anaconda3/lib/python3.8/site-packages (from keras) (1.18.5)
Requirement already satisfied: six in /home/ramesh/anaconda3/lib/python3.8/site-packages (from h5py->keras) (1.15.0)
Installing collected packages: keras
Successfully installed keras-2.4.3
```

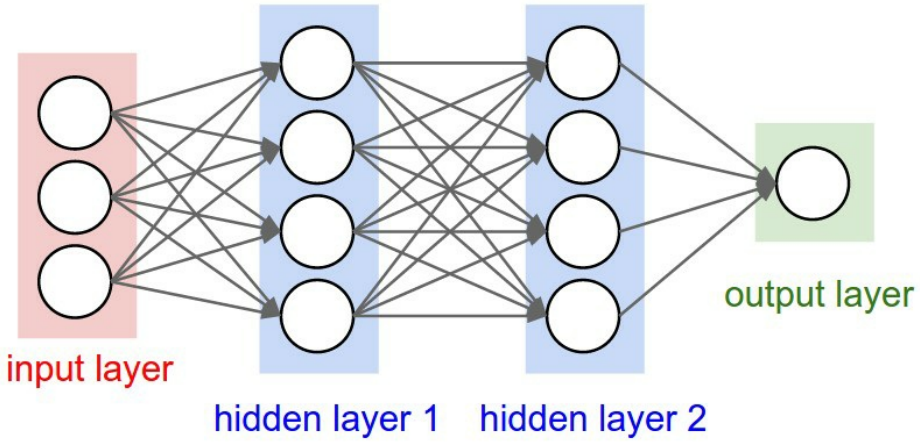
MLP Structures

- **Each MLP model is consisted of one input layer, several hidden layers, and one output layer**
- **Number of neurons in each layer is not limited**



****MLP with one hidden layer****

- **Number of input neurons: 3**
- **Number of hidden neurons: 4**
- **Number of output neurons: 2**



****MLP with two hidden layers****

- **Number of input neurons: 3**
- **Number of hidden neurons: (4, 4)**
- **Number of output neurons: 1**

(I) MLP for Regression tasks - Predict house price

- **When the target (y) is continuous (real)**
- **For loss function and evaluation metric, mean squared error (MSE) is commonly used**
- **Data:**

https://keras.io/api/datasets/boston_housing/

This is a dataset taken from the StatLib library which is maintained at Carnegie Mellon University.

Samples contain 13 attributes of houses at different locations around the Boston suburbs in the late 1970s.

Targets are the median values of the houses at a location (in k$).

The attributes themselves are defined in the StatLib website.

http://lib.stat.cmu.edu/datasets/boston

http://lib.stat.cmu.edu/datasets/boston

Variables in order:

- **CRIM per capita crime rate by town**
- **ZN proportion of residential land zoned for lots over 25,000 - sq.ft.**
- **INDUS proportion of non-retail business acres per town**
- **CHAS Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)**
- **NOX nitric oxides concentration (parts per 10 million)**
- **RM average number of rooms per dwelling**
- **AGE proportion of owner-occupied units built prior to 1940**
- **DIS weighted distances to five Boston employment centres**
- **RAD index of accessibility to radial highways**
- **TAX full-value property-tax rate per $10,000**
- **PTRATIO pupil-teacher ratio by town**
- **B 1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town**
- **LSTAT % lower status of the population**
- **MEDV Median value of owner-occupied homes in $1000's**

In [7]:

```
# Load data
from keras.datasets import boston_housing
```

In [8]:

```
(X_train, y_train), (X_test, y_test) = boston_housing.load_data()
```

In [11]:

```
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(404, 13)
(102, 13)
(404,)
(102,)
```

1. Creating a model

- **Keras model object can be created with Sequential class**
- **At the outset, the model is empty per se. It is completed by 'adding' additional layers and compilation**
- **Ref: https://keras.io/models/sequential/**
- **Ref: https://keras.io/getting-started/sequential-model-guide/**

In [12]:

```
# The Sequential model is a linear stack of layers.
# You can create a Sequential model by passing a list of
# layer instances to the constructor:

# OR

# You can also simply add layers via the .add() method:

# Ref: https://keras.io/api/models/sequential/

from keras.models import Sequential
```

In [13]:

In [13]:

```
model = Sequential() # Instantiate an empty model
```

In [15]:

```
# model.<press TAB key> # TODO: Explore its attributes
```

- **compile - First, we want to decide a model architecture, this is the number of hidden layers and activation functions, etc. (compile)**
- **fit - Secondly, we want to train our model to get all the paramters to the correct value to map our inputs to our outputs. (fit)**
- **predict - Lastly, we will want to use this model to do some feed-forward passes to predict novel inputs. (predict)**

1-1. Adding layers

- **Keras layers can be added to the model**
- **Adding layers are like stacking lego blocks one by one**
- **Doc: https://keras.io/layers/core/**

In [17]:

```
from keras.layers import Activation, Dense
```

In [18]:

```
# Method 1 - Activation will be mentioned at the time of
# layer creation

# References for Dense layer:
# https://medium.com/@hunterheidenreich/understanding-keras-dense-layers-2abadff9b990
# https://keras.io/api/layers/core_layers/dense/

model.add(Dense(10, input_shape = (13,),
                activation = 'sigmoid')
         ) # Input layer

model.add(Dense(10, activation = 'sigmoid')) # Hideen Layer1

model.add(Dense(10, activation = 'sigmoid')) # Hidden Layer2

model.add(Dense(1)) # Output Layer
```

In [8]:

```
# This is equivalent to the above code block
# Keras model with two hidden layer with 10 neurons each
# Method 2

# You should execute either previous cell or this cell,
# otherwise it will add 4 more layers

# Input layer => input_shape should be explicitly designated
#
model.add(Dense(10, input_shape = (13,)))    # A sample has 13 attributes
model.add(Activation('sigmoid'))

# Hidden layer1 => only output dimension should be designated
model.add(Dense(10))
model.add(Activation('sigmoid'))

# Hidden layer2 => only output dimension should be designated
model.add(Dense(10))
```

```
model.add(Activation('sigmoid'))

# Output layer => output dimension = 1 since it is regression problem
# In regression, it should output a single continuous value
model.add(Dense(1))
```

In [29]:

```
# Get the model configuration - Too detailed..may require time to understand
# Optional exploration..can be done after understanding MLP fully

# model.get_config()
# model.get_weights()

# Similarly we can get lot of details abot model...but we have
# trained the model yet
```

1-2. Model compile

- **Keras model should be "compiled" prior to training**
- **Types of loss (function) and optimizer should be designated**
  - **Doc (optimizers): https://keras.io/optimizers/**
  - **Doc (losses): https://keras.io/losses/**

Learning Rate

Ref: https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/

The learning rate is a hyperparameter that controls how much to change the model in response to the estimated error each time the model weights are updated. Choosing the learning rate is challenging as a value too small may result in a long training process that could get stuck, whereas a value too large may result in learning a sub-optimal set of weights too fast or an unstable training process.

In [31]:

```
# Instantiate an optimizaer
# Stochastic --> Random
# SGD: while selecting data points at each step to calculate the derivatives. SGD randomly p
icks one data point from the whole data set at each iteration to reduce the computations eno
rmously.
from keras import optimizers
sgd = optimizers.SGD(lr = 0.01)     # stochastic gradient descent optimizer
```

In [32]:

```
# Build MLP model
model.compile(optimizer = sgd,
              loss = 'mean_squared_error',
              metrics = ['mse'])  # for regression problems, mean squared error (MSE) is of
ten employed
```

Summary of the model

In [34]:

```
model.summary()
# dense - 13x10=130 weights and 10 biases, 1 bias for every neuron

# dense1 - 10 outputs from 10 neurons of dense, so,
#          10x10=100 weights and 10 biases
```

```
#           Total params = 100+10 = 110

# dense2 - 10 outputs from 10 neurons of previous dense layer
#          and dense2 has 10 neurons, all are connected
#          10x10=100 weights and 10 biases
#          Total params = 100+10 = 110

# dense3 - 10 outputs from 10 neurons of dense2 layer
#          only one neuron in this layer
#          10x1=10 weights and 1 bias for that single neuron
#          Total params = 10+1 = 11
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 10)                140
_________________________________________________________________
dense_1 (Dense)              (None, 10)                110
_________________________________________________________________
dense_2 (Dense)              (None, 10)                110
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 11
=================================================================
Total params: 371
Trainable params: 371
Non-trainable params: 0
_________________________________________________________________
```

2. Training

- **Training the model with training data provided**

In [35]:

```
# Train your MLP model
model.fit(X_train,
          y_train,
          batch_size = 50,
          epochs = 10,
          verbose = 1)

# verbose: 0, 1, or 2. Verbosity mode. 0 = silent, 1 = progress bar, 2 = one line per epoch.
Note that the progress bar is not particularly useful when logged to a file, so verbose=2 is
recommended when not running interactively (eg, in a production environment).
```

```
Epoch 1/10
9/9 [==============================] - 0s 5ms/step - loss: 327.5693 - mse: 327.5693
Epoch 2/10
9/9 [==============================] - 0s 2ms/step - loss: 94.7685 - mse: 94.7685
Epoch 3/10
9/9 [==============================] - 0s 807us/step - loss: 84.9645 - mse: 84.9645
Epoch 4/10
9/9 [==============================] - 0s 897us/step - loss: 85.5510 - mse: 85.5510
Epoch 5/10
9/9 [==============================] - 0s 1ms/step - loss: 85.0907 - mse: 85.0907
Epoch 6/10
9/9 [==============================] - 0s 829us/step - loss: 85.1794 - mse: 85.1794
Epoch 7/10
9/9 [==============================] - 0s 764us/step - loss: 85.0026 - mse: 85.0026
Epoch 8/10
9/9 [==============================] - 0s 788us/step - loss: 84.8200 - mse: 84.8200
Epoch 9/10
9/9 [==============================] - 0s 761us/step - loss: 85.1681 - mse: 85.1681
Epoch 10/10
9/9 [==============================] - 0s 980us/step - loss: 84.6657 - mse: 84.6657
```

Out[35]:

```
<tensorflow.python.keras.callbacks.History at 0x7f5663951b50>
```

3. Evaluation

- **Keras model can be evaluated with evaluate() function**
- **Evaluation results are contained in a list**
  - **Doc (metrics): https://keras.io/metrics/**

In [37]:

```
results = model.evaluate(X_test, y_test)
```

```
4/4 [==============================] - 0s 619us/step - loss: 83.1120 - mse: 83.1120
```

In [38]:

```
print(model.metrics_names)      # list of metric names the model is employing
print(results)                  # actual figure of metrics computed
```

```
['loss', 'mse']
[83.11197662353516, 83.11197662353516]
```

In [39]:

```
print('loss: ', results[0])
print('mse: ', results[1])
```

```
loss:  83.11197662353516
mse:  83.11197662353516
```

(II) MLP for classification tasks

- **When the target (y) is discrete (categorical)**
- **For loss function, cross-entropy is used and for evaluation metric, accuracy is commonly used**

In [70]:

```
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
```

In [71]:

```
whole_data = load_breast_cancer()
```

In [72]:

```
X_data = whole_data.data
y_data = whole_data.target
```

In [75]:

```
#y_data
```

In [73]:

```
X_train, X_test, y_train, y_test = train_test_split(X_data, y_data, test_size = 0.3, random
_state = 7)
```

Dataset Description

- **Breast cancer dataset has total 569 data instances (212 malign, 357 benign instances)**
- **30 attributes (features) to predict the binary class (M/B)**
- **Doc: http://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_breast_cancer.html#sklearn.datasets.load_breast_can**

In [76]:

```
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(398, 30)
(171, 30)
(398,)
(171,)
```

1. Creating a model

- **Same with regression model at the outset**

In [77]:

```
from keras.models import Sequential
```

In [78]:

```
model = Sequential()
```

1-1. Adding layers

- **Keras layers can be added to the model**
- **Adding layers are like stacking lego blocks one by one**
- **It should be noted that as this is a classification problem, sigmoid layer (softmax for multi-class problems) should be added**
- **Doc: https://keras.io/layers/core/**

In [80]:

```
# Method 1: This is equivalent to the below code block
model.add(Dense(10, input_shape = (30,), activation = 'sigmoid'))
model.add(Dense(10, activation = 'sigmoid'))
model.add(Dense(10, activation = 'sigmoid'))
model.add(Dense(2, activation = 'softmax'))
```

In [ ]:

```
# Method 2: Keras model with two hidden layer with 10 neurons each
model.add(Dense(10, input_shape = (30,)))    # Input layer => input_shape should be explici
tly designated
model.add(Activation('sigmoid'))
model.add(Dense(10))                         # Hidden layer => only output dimension should
be designated
model.add(Activation('sigmoid'))
model.add(Dense(10))                         # Hidden layer => only output dimension should
be designated
model.add(Activation('sigmoid'))
model.add(Dense(2))                          # Output layer => output dimension = 2 since i
```

```
t is binary classification problem
model.add(Activation('softmax'))
```

1-2. Model compile

- **Keras model should be "compiled" prior to training**
- **Types of loss (function) and optimizer should be designated**
  - **Doc (optimizers): https://keras.io/optimizers/**
  - **Doc (losses): https://keras.io/losses/**

In [81]:

```
from keras import optimizers
```

In [82]:

```
sgd = optimizers.SGD(lr = 0.01)     # stochastic gradient descent optimizer
```

In [83]:

```
model.compile(optimizer = sgd,
              loss = 'binary_crossentropy',
              metrics = ['accuracy'])
```

Summary of the model

In [84]:

```
model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_20 (Dense)             (None, 10)                310
_________________________________________________________________
dense_21 (Dense)             (None, 10)                110
_________________________________________________________________
dense_22 (Dense)             (None, 10)                110
_________________________________________________________________
dense_23 (Dense)             (None, 2)                 22
_________________________________________________________________
dense_24 (Dense)             (None, 10)                30
_________________________________________________________________
dense_25 (Dense)             (None, 10)                110
_________________________________________________________________
dense_26 (Dense)             (None, 10)                110
_________________________________________________________________
dense_27 (Dense)             (None, 2)                 22
=================================================================
Total params: 824
Trainable params: 824
Non-trainable params: 0
_________________________________________________________________
```

2. Training

- **Training the model with training data provided**

In [85]:

```
model.fit(X_train,
          y_train,
          batch_size = 50,
          epochs = 100,
          verbose = 1)
```

```
Epoch 1/100
8/8 [==============================] - 0s 1ms/step - loss: 0.6970 - accuracy: 0.6055
Epoch 2/100
8/8 [==============================] - 0s 1ms/step - loss: 0.6959 - accuracy: 0.6055
Epoch 3/100
8/8 [==============================] - 0s 1ms/step - loss: 0.6951 - accuracy: 0.6055
Epoch 4/100
8/8 [==============================] - 0s 1ms/step - loss: 0.6945 - accuracy: 0.6055
Epoch 5/100
8/8 [==============================] - 0s 2ms/step - loss: 0.6941 - accuracy: 0.6055
Epoch 6/100
8/8 [==============================] - 0s 2ms/step - loss: 0.6938 - accuracy: 0.6055
Epoch 7/100
8/8 [==============================] - 0s 1ms/step - loss: 0.6936 - accuracy: 0.6055
Epoch 8/100
8/8 [==============================] - 0s 1ms/step - loss: 0.6935 - accuracy: 0.6055
Epoch 9/100
8/8 [==============================] - 0s 1ms/step - loss: 0.6934 - accuracy: 0.6055
Epoch 10/100
8/8 [==============================] - 0s 1ms/step - loss: 0.6933 - accuracy: 0.6055
Epoch 11/100
8/8 [==============================] - 0s 1ms/step - loss: 0.6933 - accuracy: 0.6055
Epoch 12/100
8/8 [==============================] - 0s 1ms/step - loss: 0.6932 - accuracy: 0.6055
Epoch 13/100
8/8 [==============================] - 0s 1ms/step - loss: 0.6932 - accuracy: 0.6055
Epoch 14/100
8/8 [==============================] - 0s 1ms/step - loss: 0.6932 - accuracy: 0.6055
Epoch 15/100
8/8 [==============================] - 0s 2ms/step - loss: 0.6932 - accuracy: 0.6055
Epoch 16/100
8/8 [==============================] - 0s 2ms/step - loss: 0.6932 - accuracy: 0.6055
Epoch 17/100
8/8 [==============================] - 0s 2ms/step - loss: 0.6932 - accuracy: 0.6055
Epoch 18/100
8/8 [==============================] - 0s 890us/step - loss: 0.6932 - accuracy: 0.6055
Epoch 19/100
8/8 [==============================] - 0s 1ms/step - loss: 0.6932 - accuracy: 0.6055
Epoch 20/100
8/8 [==============================] - 0s 1ms/step - loss: 0.6932 - accuracy: 0.6055
Epoch 21/100
8/8 [==============================] - 0s 1ms/step - loss: 0.6932 - accuracy: 0.6055
Epoch 22/100
8/8 [==============================] - 0s 1ms/step - loss: 0.6931 - accuracy: 0.6055
Epoch 23/100
8/8 [==============================] - 0s 1ms/step - loss: 0.6931 - accuracy: 0.6055
Epoch 24/100
8/8 [==============================] - 0s 2ms/step - loss: 0.6931 - accuracy: 0.6055
Epoch 25/100
8/8 [==============================] - 0s 1ms/step - loss: 0.6931 - accuracy: 0.6055
Epoch 26/100
8/8 [==============================] - 0s 993us/step - loss: 0.6931 - accuracy: 0.6055
Epoch 27/100
8/8 [==============================] - 0s 1ms/step - loss: 0.6931 - accuracy: 0.6055
Epoch 28/100
8/8 [==============================] - 0s 1ms/step - loss: 0.6931 - accuracy: 0.6055
Epoch 29/100
8/8 [==============================] - 0s 850us/step - loss: 0.6931 - accuracy: 0.6055
Epoch 30/100
8/8 [==============================] - 0s 923us/step - loss: 0.6931 - accuracy: 0.6055
Epoch 31/100
8/8 [==============================] - 0s 2ms/step - loss: 0.6931 - accuracy: 0.6055
```

```
Epoch 32/100
8/8 [==============================] - 0s 1ms/step - loss: 0.6931 - accuracy: 0.6055
Epoch 33/100
8/8 [==============================] - 0s 1ms/step - loss: 0.6931 - accuracy: 0.6055
Epoch 34/100
8/8 [==============================] - 0s 991us/step - loss: 0.6931 - accuracy: 0.6055
Epoch 35/100
8/8 [==============================] - 0s 1ms/step - loss: 0.6931 - accuracy: 0.6055
Epoch 36/100
8/8 [==============================] - 0s 1ms/step - loss: 0.6931 - accuracy: 0.6055
Epoch 37/100
8/8 [==============================] - 0s 1ms/step - loss: 0.6931 - accuracy: 0.6055
Epoch 38/100
8/8 [==============================] - 0s 986us/step - loss: 0.6931 - accuracy: 0.6055
Epoch 39/100
8/8 [==============================] - 0s 1ms/step - loss: 0.6931 - accuracy: 0.6055
Epoch 40/100
8/8 [==============================] - 0s 1ms/step - loss: 0.6931 - accuracy: 0.6055
Epoch 41/100
8/8 [==============================] - 0s 999us/step - loss: 0.6931 - accuracy: 0.6055
Epoch 42/100
8/8 [==============================] - 0s 1ms/step - loss: 0.6931 - accuracy: 0.6055
Epoch 43/100
8/8 [==============================] - 0s 1ms/step - loss: 0.6931 - accuracy: 0.6055
Epoch 44/100
8/8 [==============================] - 0s 965us/step - loss: 0.6931 - accuracy: 0.6055
Epoch 45/100
8/8 [==============================] - 0s 890us/step - loss: 0.6931 - accuracy: 0.6080
Epoch 46/100
8/8 [==============================] - 0s 1ms/step - loss: 0.6931 - accuracy: 0.6457
Epoch 47/100
8/8 [==============================] - 0s 932us/step - loss: 0.6931 - accuracy: 0.6533
Epoch 48/100
8/8 [==============================] - 0s 1ms/step - loss: 0.6931 - accuracy: 0.6633
Epoch 49/100
8/8 [==============================] - 0s 1ms/step - loss: 0.6931 - accuracy: 0.6633
Epoch 50/100
8/8 [==============================] - 0s 972us/step - loss: 0.6931 - accuracy: 0.6658
Epoch 51/100
8/8 [==============================] - 0s 1ms/step - loss: 0.6931 - accuracy: 0.6683
Epoch 52/100
8/8 [==============================] - 0s 1ms/step - loss: 0.6931 - accuracy: 0.6683
Epoch 53/100
8/8 [==============================] - 0s 1ms/step - loss: 0.6931 - accuracy: 0.6683
Epoch 54/100
8/8 [==============================] - 0s 943us/step - loss: 0.6931 - accuracy: 0.6683
Epoch 55/100
8/8 [==============================] - 0s 1ms/step - loss: 0.6931 - accuracy: 0.6709
Epoch 56/100
8/8 [==============================] - 0s 957us/step - loss: 0.6931 - accuracy: 0.6709
Epoch 57/100
8/8 [==============================] - 0s 1ms/step - loss: 0.6931 - accuracy: 0.6734
Epoch 58/100
8/8 [==============================] - 0s 1ms/step - loss: 0.6931 - accuracy: 0.6734
Epoch 59/100
8/8 [==============================] - 0s 1ms/step - loss: 0.6931 - accuracy: 0.6734
Epoch 60/100
8/8 [==============================] - 0s 1ms/step - loss: 0.6931 - accuracy: 0.6734
Epoch 61/100
8/8 [==============================] - 0s 959us/step - loss: 0.6931 - accuracy: 0.6734
Epoch 62/100
8/8 [==============================] - 0s 968us/step - loss: 0.6931 - accuracy: 0.6734
Epoch 63/100
8/8 [==============================] - 0s 1ms/step - loss: 0.6931 - accuracy: 0.6734
Epoch 64/100
8/8 [==============================] - 0s 1ms/step - loss: 0.6931 - accuracy: 0.6734
Epoch 65/100
8/8 [==============================] - 0s 961us/step - loss: 0.6931 - accuracy: 0.6734
```

```
Epoch 66/100
8/8 [==============================] - 0s 996us/step - loss: 0.6931 - accuracy: 0.6734
Epoch 67/100
8/8 [==============================] - 0s 872us/step - loss: 0.6931 - accuracy: 0.6734
Epoch 68/100
8/8 [==============================] - 0s 1ms/step - loss: 0.6931 - accuracy: 0.6734
Epoch 69/100
8/8 [==============================] - 0s 1ms/step - loss: 0.6931 - accuracy: 0.6734
Epoch 70/100
8/8 [==============================] - 0s 1ms/step - loss: 0.6931 - accuracy: 0.6734
Epoch 71/100
8/8 [==============================] - 0s 1ms/step - loss: 0.6931 - accuracy: 0.6709
Epoch 72/100
8/8 [==============================] - 0s 1ms/step - loss: 0.6931 - accuracy: 0.6709
Epoch 73/100
8/8 [==============================] - 0s 1ms/step - loss: 0.6931 - accuracy: 0.6734
Epoch 74/100
8/8 [==============================] - 0s 940us/step - loss: 0.6931 - accuracy: 0.6734
Epoch 75/100
8/8 [==============================] - 0s 945us/step - loss: 0.6931 - accuracy: 0.6734
Epoch 76/100
8/8 [==============================] - 0s 1ms/step - loss: 0.6931 - accuracy: 0.6734
Epoch 77/100
8/8 [==============================] - 0s 2ms/step - loss: 0.6931 - accuracy: 0.6734
Epoch 78/100
8/8 [==============================] - 0s 1ms/step - loss: 0.6931 - accuracy: 0.6734
Epoch 79/100
8/8 [==============================] - 0s 1ms/step - loss: 0.6931 - accuracy: 0.6734
Epoch 80/100
8/8 [==============================] - 0s 1ms/step - loss: 0.6931 - accuracy: 0.6734
Epoch 81/100
8/8 [==============================] - 0s 1ms/step - loss: 0.6931 - accuracy: 0.6759
Epoch 82/100
8/8 [==============================] - 0s 1ms/step - loss: 0.6931 - accuracy: 0.6759
Epoch 83/100
8/8 [==============================] - 0s 838us/step - loss: 0.6931 - accuracy: 0.6759
Epoch 84/100
8/8 [==============================] - 0s 1ms/step - loss: 0.6931 - accuracy: 0.6759
Epoch 85/100
8/8 [==============================] - 0s 1ms/step - loss: 0.6931 - accuracy: 0.6759
Epoch 86/100
8/8 [==============================] - 0s 874us/step - loss: 0.6931 - accuracy: 0.6759
Epoch 87/100
8/8 [==============================] - 0s 1ms/step - loss: 0.6931 - accuracy: 0.6759
Epoch 88/100
8/8 [==============================] - 0s 970us/step - loss: 0.6931 - accuracy: 0.6759
Epoch 89/100
8/8 [==============================] - 0s 1ms/step - loss: 0.6931 - accuracy: 0.6759
Epoch 90/100
8/8 [==============================] - 0s 1ms/step - loss: 0.6931 - accuracy: 0.6759
Epoch 91/100
8/8 [==============================] - 0s 952us/step - loss: 0.6931 - accuracy: 0.6759
Epoch 92/100
8/8 [==============================] - 0s 884us/step - loss: 0.6931 - accuracy: 0.6759
Epoch 93/100
8/8 [==============================] - 0s 1ms/step - loss: 0.6931 - accuracy: 0.6759
Epoch 94/100
8/8 [==============================] - 0s 944us/step - loss: 0.6931 - accuracy: 0.6759
Epoch 95/100
8/8 [==============================] - 0s 1ms/step - loss: 0.6931 - accuracy: 0.6759
Epoch 96/100
8/8 [==============================] - 0s 1ms/step - loss: 0.6931 - accuracy: 0.6759
Epoch 97/100
8/8 [==============================] - 0s 1ms/step - loss: 0.6931 - accuracy: 0.6759
Epoch 98/100
8/8 [==============================] - 0s 876us/step - loss: 0.6931 - accuracy: 0.6759
Epoch 99/100
```

```
8/8 [==============================] - 0s 1ms/step - loss: 0.6931 - accuracy: 0.6759
Epoch 100/100
8/8 [==============================] - 0s 1ms/step - loss: 0.6931 - accuracy: 0.6759
```

Out[85]:

```
<tensorflow.python.keras.callbacks.History at 0x7f56181d8ee0>
```

3. Evaluation

- **Keras model can be evaluated with evaluate() function**
- **Evaluation results are contained in a list**
  - **Doc (metrics): https://keras.io/metrics/**

In [86]:

```
results = model.evaluate(X_test, y_test)
```

```
6/6 [==============================] - 0s 1ms/step - loss: 0.6931 - accuracy: 0.7602
```

In [87]:

```
print(model.metrics_names)      # list of metric names the model is employing
print(results)                  # actual figure of metrics computed
```

```
['loss', 'accuracy']
[0.6931469440460205, 0.7602339386940002]
```

In [88]:

```
print('loss: ', results[0])
print('accuracy: ', results[1])
```

```
loss:  0.6931469440460205
accuracy:  0.7602339386940002
```