```
#CNN (A)

from keras.datasets import imdb
from keras.preprocessing import sequence
max_features = 10000
maxlen = 100
print('Loading data...')
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=max_features)
print(len(x_train), 'train sequences')
print(len(x_test), 'test sequences')
print('Pad sequences (samples x time)')
x_train = sequence.pad_sequences(x_train, maxlen=maxlen)
x_test = sequence.pad_sequences(x_test, maxlen=maxlen)
print('x_train shape:', x_train.shape)
print('x_test shape:', x_test.shape)
```

```
Loading data...
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb.npz
17464789/17464789 [==============================] - 1s 0us/step
25000 train sequences
25000 test sequences
Pad sequences (samples x time)
x_train shape: (25000, 100)
x_test shape: (25000, 100)
```

```
#CNN (B)

from keras.datasets import imdb
from keras.models import Sequential
from keras.layers import Embedding, Conv1D, GlobalMaxPooling1D, Dense
from keras.preprocessing import sequence
max_features = 10000
maxlen = 100
print('Loading data...')
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=max_features)
print(len(x_train), 'train sequences')
print(len(x_test), 'test sequences')
print('Pad sequences (samples x time)')
x_train = sequence.pad_sequences(x_train, maxlen=maxlen)
x_test = sequence.pad_sequences(x_test, maxlen=maxlen)
print('x_train shape:', x_train.shape)
print('x_test shape:', x_test.shape)
model = Sequential()
model.add(Embedding(max_features, 128, input_length=maxlen))
model.add(Conv1D(64, 5, activation='relu'))
model.add(GlobalMaxPooling1D())
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
print('Training model...')
model.fit(x_train, y_train, epochs=3, batch_size=32, validation_split=0.2)
print('Evaluating model...')
loss, accuracy = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', loss)
print('Test accuracy:', accuracy)
```

```
Loading data...
25000 train sequences
25000 test sequences
Pad sequences (samples x time)
x_train shape: (25000, 100)
x_test shape: (25000, 100)
Training model...
Epoch 1/3
625/625 [==============================] - 30s 47ms/step - loss: 0.4747 - accuracy: 0.7708 - val_loss: 0.3508 - val_accuracy: 0.841
Epoch 2/3
625/625 [==============================] - 29s 47ms/step - loss: 0.2576 - accuracy: 0.8953 - val_loss: 0.3365 - val_accuracy: 0.854
Epoch 3/3
625/625 [==============================] - 31s 49ms/step - loss: 0.1238 - accuracy: 0.9590 - val_loss: 0.3844 - val_accuracy: 0.849
Evaluating model...
Test loss: 0.3903481364250183
Test accuracy: 0.8460400104522705
```

```
# RNN sentiment analysis on movie reviews

from keras.datasets import imdb
from keras.preprocessing.text import Tokenizer
from keras.utils import pad_sequences
from keras import Sequential
from keras.layers import Dense, SimpleRNN, Embedding, Flatten

# Load IMDb dataset
(X_train, y_train), (X_test, y_test) = imdb.load_data(num_words=10000)

# Pad sequences
X_train = pad_sequences(X_train, padding='post', maxlen=50)
X_test = pad_sequences(X_test, padding='post', maxlen=50)

# Define model
model = Sequential()

# Embedding layer
model.add(Embedding(10000, 32, input_length=50))

# SimpleRNN layer
model.add(SimpleRNN(32))

# Dense layer
model.add(Dense(1, activation='sigmoid'))

# Compile model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Print model summary
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_1 (Embedding)     (None, 50, 32)            320000

 simple_rnn_1 (SimpleRNN)    (None, 32)                2080

 dense_1 (Dense)             (None, 1)                 33

=================================================================
Total params: 322113 (1.23 MB)
Trainable params: 322113 (1.23 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
# ADD TWO NUMBERS

#Step-1: Import
import numpy as np
import tensorflow as tf
from random import randrange
#Step-2: Generate Training Data
trainingInput = [[i, i + randrange(5000)] for i in range(1, 5000)]
trainingOutput = [(input [0] + input [1]) for input in trainingInput ]
testInput = [[5, 5], [1, 9], [2, 5], [6, 3], [1, 4]]
testOutput = [10, 10, 7, 9, 5]
#Step-3: Build the model
model = tf.keras.Sequential()
model.add(tf.keras.layers.Flatten(input_shape=(2,)))
model.add(tf.keras.layers.Dense(64, activation=tf.nn.relu))
model.add(tf.keras.layers.Dense(64, activation=tf.nn.relu))
model.add(tf.keras.layers.Dense(1))
#Step-4: Compile the model
model.compile(optimizer='adam', loss=tf.keras.losses.mae, metrics=['mae'])
#Step-5: Training
model.fit(trainingInput, trainingOutput, batch_size=5, epochs=3)
#Step-6: Final Evaluation and Prediction
test_loss, test_acc = model.evaluate(testInput, testOutput)
print("Test Accuracy : ", test_acc)
a = np.array([[3, 3000], [4, 5], [1,10], [2,10],[5,9], [4,10], [1,15]])
print(model.predict(a))
```

```
Epoch 1/3
1000/1000 [==============================] - 5s 4ms/step - loss: 241.3566 - mae: 241.3566
Epoch 2/3
1000/1000 [==============================] - 9s 9ms/step - loss: 28.6675 - mae: 28.6675
```

```
Epoch 3/3
1000/1000 [==============================] - 5s 5ms/step - loss: 28.5529 - mae: 28.5529
1/1 [==============================] - 1s 581ms/step - loss: 0.2576 - mae: 0.2576
Test Accuracy :  0.25757989287376404
1/1 [==============================] - 0s 175ms/step
[[3010.5828  ]
 [   9.251566]
 [  11.226904]
 [  12.218241]
 [  14.256031]
 [  14.250777]
 [  16.240612]]
```

```
#ANN
from keras.models import Sequential
from keras.layers import Dense,Activation
import numpy as np
import pandas as pd
from sklearn import datasets
iris =datasets.load_iris()
X, y = datasets.load_iris( return_X_y = True)
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.40)# Define the network model and its arguments.
# Set the number of neurons/nodes for each layer:
model = Sequential()
model.add(Dense(2,input_shape=(4,)))
model.add(Activation('sigmoid'))
model.add(Dense(1))
model.add(Activation('sigmoid'))
#sgd = SGD(lr=0.0001, decay=1e-6, momentum=0.9, nesterov=True)
#model.compile(loss='categorical_crossentropy', optimizer=sgd,
metrics=['accuracy']# Compile the model and calculate its accuracy:
model.compile(loss='mean_squared_error', optimizer='sgd',metrics=['accuracy']) #model.fit(X_train, y_train, batch_size=32,epochs=3)
# Print a summary of the Keras model:
model.summary()
#model.fit(X_train, y_train)
#model.fit(X_train, y_train,batch_size=32, epochs=3)
model.fit(X_train, y_train, epochs=5)
score = model.evaluate(X_test, y_test)
print(score)
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_2 (Dense)             (None, 2)                 10

 activation (Activation)     (None, 2)                 0

 dense_3 (Dense)             (None, 1)                 3

 activation_1 (Activation)   (None, 1)                 0

=================================================================
Total params: 13 (52.00 Byte)
Trainable params: 13 (52.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/5
3/3 [==============================] - 1s 6ms/step - loss: 0.6789 - accuracy: 0.3333
Epoch 2/5
3/3 [==============================] - 0s 4ms/step - loss: 0.6782 - accuracy: 0.3333
Epoch 3/5
3/3 [==============================] - 0s 4ms/step - loss: 0.6775 - accuracy: 0.3333
Epoch 4/5
3/3 [==============================] - 0s 4ms/step - loss: 0.6769 - accuracy: 0.3333
Epoch 5/5
3/3 [==============================] - 0s 4ms/step - loss: 0.6762 - accuracy: 0.3333
2/2 [==============================] - 0s 8ms/step - loss: 0.7190 - accuracy: 0.3333
[0.7189559936523438, 0.3333333432674408]
```

```
#X-OR GATE

# importing Python library
import numpy as np

# define Unit Step Function
def unitStep(v):
    if v >= 0:
        return 1
    else:
        return 0

# design Perceptron Model
def perceptronModel(x, w, b):
    v = np.dot(w, x) + b
    y = unitStep(v)
    return y

# NOT Logic Function
# wNOT = -1, bNOT = 0.5
def NOT_logicFunction(x):
    wNOT = -1
    bNOT = 0.5
    return perceptronModel(x, wNOT, bNOT)

# AND Logic Function
# here w1 = wAND1 = 1,
# w2 = wAND2 = 1, bAND = -1.5
def AND_logicFunction(x):
    w = np.array([1, 1])
    bAND = -1.5
    return perceptronModel(x, w, bAND)

# OR Logic Function
# w1 = 1, w2 = 1, bOR = -0.5
def OR_logicFunction(x):
    w = np.array([1, 1])
    bOR = -0.5
    return perceptronModel(x, w, bOR)

# XOR Logic Function
# with AND, OR and NOT
# function calls in sequence
def XOR_logicFunction(x):
    y1 = AND_logicFunction(x)
    y2 = OR_logicFunction(x)
    y3 = NOT_logicFunction(y1)
    final_x = np.array([y2, y3])
    finalOutput = AND_logicFunction(final_x)
    return finalOutput

# testing the Perceptron Model
test1 = np.array([0, 1])
test2 = np.array([1, 1])
test3 = np.array([0, 0])
test4 = np.array([1, 0])

print("XOR({}, {}) = {}".format(0, 1, XOR_logicFunction(test1)))
print("XOR({}, {}) = {}".format(1, 1, XOR_logicFunction(test2)))
print("XOR({}, {}) = {}".format(0, 0, XOR_logicFunction(test3)))
print("XOR({}, {}) = {}".format(1, 0, XOR_logicFunction(test4)))
```

```
XOR(0, 1) = 1
XOR(1, 1) = 0
XOR(0, 0) = 0
XOR(1, 0) = 1
```