Question 1.

```
#importing the required modules.
import keras
import numpy as np
from keras.models import Sequential
from keras.layers import Dense, Activation
X = np.random.randint(10, size=(20,1))


y = np.random.randint(2, size=(20,1))




model = Sequential()
model.add(Dense(4, input_dim=1, activation='relu'))
model.add(Dense(2, input_dim=1, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
#printing the model summary
print(model.summary())
model.compile(loss="mean_squared_error", optimizer="adam")

model.fit(X, y, epochs=50)
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_9 (Dense)             (None, 4)                 8

 dense_10 (Dense)            (None, 2)                 10

 dense_11 (Dense)            (None, 1)                 3

=================================================================
Total params: 21
Trainable params: 21
Non-trainable params: 0
_________________________________________________________________
None
Epoch 1/50
1/1 [==============================] - 0s 164ms/step - loss: 0.2500
Epoch 2/50
1/1 [==============================] - 0s 3ms/step - loss: 0.2499
Epoch 3/50
1/1 [==============================] - 0s 2ms/step - loss: 0.2498
```

```
Epoch 4/50
1/1 [==============================] - 0s 3ms/step - loss: 0.2497
Epoch 5/50
1/1 [==============================] - 0s 4ms/step - loss: 0.2496
Epoch 6/50
1/1 [==============================] - 0s 2ms/step - loss: 0.2495
Epoch 7/50
1/1 [==============================] - 0s 2ms/step - loss: 0.2494
Epoch 8/50
1/1 [==============================] - 0s 3ms/step - loss: 0.2493
Epoch 9/50
1/1 [==============================] - 0s 3ms/step - loss: 0.2492
Epoch 10/50
1/1 [==============================] - 0s 2ms/step - loss: 0.2491
Epoch 11/50
1/1 [==============================] - 0s 2ms/step - loss: 0.2490
Epoch 12/50
1/1 [==============================] - 0s 3ms/step - loss: 0.2489
Epoch 13/50
1/1 [==============================] - 0s 4ms/step - loss: 0.2488
Epoch 14/50
1/1 [==============================] - 0s 3ms/step - loss: 0.2487
Epoch 15/50
1/1 [==============================] - 0s 2ms/step - loss: 0.2486
Epoch 16/50
1/1 [==============================] - 0s 3ms/step - loss: 0.2485
Epoch 17/50
1/1 [==============================] - 0s 3ms/step - loss: 0.2484
Epoch 18/50
1/1 [==============================] - 0s 3ms/step - loss: 0.2483
Epoch 19/50
1/1 [==============================] - 0s 3ms/step - loss: 0.2482
Epoch 20/50
1/1 [==============================] - 0s 2ms/step - loss: 0.2481
Epoch 21/50
1/1 [==============================] - 0s 3ms/step - loss: 0.2480
Epoch 22/50
1/1 [==============================] - 0s 3ms/step - loss: 0.2479
Epoch 23/50
1/1 [==============================] - 0s 3ms/step - loss: 0.2478
Epoch 24/50
1/1 [==============================] - 0s 3ms/step - loss: 0.2477
Epoch 25/50
1/1 [==============================] - 0s 2ms/step - loss: 0.2476
Epoch 26/50
1/1 [==============================] - 0s 3ms/step - loss: 0.2475
Epoch 27/50
1/1 [==============================] - 0s 2ms/step - loss: 0.2474
Epoch 28/50
1/1 [==============================] - 0s 2ms/step - loss: 0.2474
```

```
Epoch 29/50
1/1 [==============================] - 0s 3ms/step - loss: 0.2473
Epoch 30/50
1/1 [==============================] - 0s 3ms/step - loss: 0.2472
Epoch 31/50
1/1 [==============================] - 0s 2ms/step - loss: 0.2471
Epoch 32/50
1/1 [==============================] - 0s 4ms/step - loss: 0.2470
Epoch 33/50
1/1 [==============================] - 0s 2ms/step - loss: 0.2469
Epoch 34/50
1/1 [==============================] - 0s 3ms/step - loss: 0.2468
Epoch 35/50
1/1 [==============================] - 0s 3ms/step - loss: 0.2467
Epoch 36/50
1/1 [==============================] - 0s 3ms/step - loss: 0.2466
Epoch 37/50
1/1 [==============================] - 0s 2ms/step - loss: 0.2465
Epoch 38/50
1/1 [==============================] - 0s 2ms/step - loss: 0.2464
Epoch 39/50
1/1 [==============================] - 0s 2ms/step - loss: 0.2463
Epoch 40/50
1/1 [==============================] - 0s 2ms/step - loss: 0.2462
Epoch 41/50
1/1 [==============================] - 0s 1ms/step - loss: 0.2461
Epoch 42/50
1/1 [==============================] - 0s 1000us/step - loss: 0.2460
Epoch 43/50
1/1 [==============================] - 0s 2ms/step - loss: 0.2459
Epoch 44/50
1/1 [==============================] - 0s 2ms/step - loss: 0.2458
Epoch 45/50
1/1 [==============================] - 0s 4ms/step - loss: 0.2457
Epoch 46/50
1/1 [==============================] - 0s 2ms/step - loss: 0.2457
Epoch 47/50
1/1 [==============================] - 0s 2ms/step - loss: 0.2456
Epoch 48/50
1/1 [==============================] - 0s 2ms/step - loss: 0.2455
Epoch 49/50
1/1 [==============================] - 0s 1ms/step - loss: 0.2454
Epoch 50/50
1/1 [==============================] - 0s 2ms/step - loss: 0.2453

<keras.callbacks.History at 0x17c55ca4e50>
```

Question 2.

```
#importing all the packages
import sys
```

```
#importing computer vision package
import cv2
import numpy as np


# Grayscale Image
def processImage(image):
    #reading the image
    image = cv2.imread(image)
    image = cv2.cvtColor(src=image, code=cv2.COLOR_BGR2GRAY)
    return image


def convolve2D(image, kernel, padding=0, strides=1):
    # Cross Correlation
    kernel = np.flipud(np.fliplr(kernel))
    print('kernel = ', kernel)
    print(kernel.shape)

    # Gather Shapes of Kernel + Image + Padding
    xKernShape = kernel.shape[0]
    yKernShape = kernel.shape[1]
    xImgShape = 10
    yImgShape = 10

    # Shape of Output Convolution
    xOutput = int(((xImgShape - xKernShape + 2 * padding) / strides) +
1)
    yOutput = int(((yImgShape - yKernShape + 2 * padding) / strides) +
1)
    output = np.zeros((xOutput, yOutput))

    # Apply Equal Padding to All Sides
    if padding != 0:
        imagePadded = np.zeros((10 + padding*2, 10 + padding*2))
        imagePadded[int(padding):int(-1 * padding), int(padding):int(-
1 * padding)] = image
    else:
        imagePadded = image

    # Iterate through image
    for y in range(10):
        # Exit Convolution
        if y > 10 - yKernShape:
            break
        # Only Convolve if y has gone down by the specified Strides
        if y % strides == 0:
            for x in range(10):
                # Go to next row once kernel is out of bounds
```

```
                if x > 10 - xKernShape:
                    break
                try:
                    # Only Convolve if x has moved by the specified
Strides
                    if x % strides == 0:
                        output[x, y] = (kernel * imagePadded[x: x +
xKernShape, y: y + yKernShape]).sum()
                except:
                    break

    return output


if __name__ == '__main__':
    # Grayscale Image for processing.
    image = processImage('px10.jpg')
    print('image = ', image)
    print(image.shape)

    #considering various values of kernal and 0bserving the result.

    # Edge Detection Kernel
    kernel = np.array([[-1, -1, -1], [-1, 8, -1], [-1, -1, -1]])

    # Convolve and Save Output
    output = convolve2D(image, kernel, padding=2)
    x=cv2.imwrite('2Dconvolved-image-1.jpg', output)
    if(x):
        print("Image saved as 2Dconvolved-image-1.jpg in the source
folder.")


     # Edge Detection Kernel
    kernel = np.array([[-1, -1, -1], [-1, 10, -1], [-1, -1, -1]])

    # Convolve and Save Output
    output1 = convolve2D(image, kernel, padding=2)
    x=cv2.imwrite('2Dconvolved-image-2.jpg', output1)
    if(x):
        print("Image saved as 2Dconvolved-image-2.jpg in the source
folder.")

    # Edge Detection Kernel
    kernel = np.array([[-1, -1, -1], [-1, 15, -1], [-1, -1, -1]])

    # Convolve and Save Output
    output2 = convolve2D(image, kernel, padding=2)
    x=cv2.imwrite('2Dconvolved-image-3.jpg', output2)
    if(x):
```

```
        print("Image saved as 2Dconvolved-image-3.jpg in the source
folder.")
```

```
image =  [[ 78  79  24  18   0  57  44  36  70  29]
 [139  86  63   4  22  13  18  40  69  55]
 [136 125  75  33  17   0   2  32  54  74]
 [ 55 141  96  64  25   1  21   9  27  62]
 [  5  92 149  73  64   9  27   0  12  28]
 [ 17  40 146 109  74  50   0  15  11   5]
 [ 28  21  78 141  82  82  11  19   9   8]
 [ 18  12  27 125 121  67  77   1   3  22]
 [  4  25  25  40 165  82  75  57   4   5]
 [ 38  25  10   7  69 151  98  58   3   5]]
(10, 10)
kernel =  [[-1 -1 -1]
 [-1  8 -1]
 [-1 -1 -1]]
(3, 3)
Image saved as 2Dconvolved-image-1.jpg in the source folder.
kernel =  [[-1 -1 -1]
 [-1 10 -1]
 [-1 -1 -1]]
(3, 3)
Image saved as 2Dconvolved-image-2.jpg in the source folder.
kernel =  [[-1 -1 -1]
 [-1 15 -1]
 [-1 -1 -1]]
(3, 3)
Image saved as 2Dconvolved-image-3.jpg in the source folder.
```

Question 3 (d).

```
#importing numpy
import numpy as np
# import tensorflow as tf
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()
```

```
#from nltk.tokenize import word_tokenize



corpus_raw = 'The bench also accepted the child immunization policy,
but stipulated that the clinical study data be made public as soon as
possible.'
# convert to lower case
corpus_raw = corpus_raw.lower()
corpus_raw = corpus_raw.replace(",", "")
words = []
for word in corpus_raw.split():
     # because we don't want to treat . as a word
    if word != '.':
        words.append(word.replace(".", ""))
#all duplicate words are removed
words = set(words)

# words = word_tokenize(corpus_raw)
word2int = {}
int2word = {}
vocab_size = len(words) # gives the total number of unique words
print('Vocab size: ', vocab_size)
print('Words:', words)

for i,word in enumerate(words):
    word2int[word] = i
    int2word[i] = word


# raw sentences is a list of sentences.
raw_sentences = corpus_raw.split('.')
sentences = []

print( 'Total number of sentenses are:', len(raw_sentences))
for i, sentence in enumerate(raw_sentences):
        sentences.append(sentence.split())
#        print(i , ':', sentence.split())

#using fixed window size, pairs of word are created for input
WINDOW_SIZE = 2

data = []
for sentence in sentences:
    for word_index, word in enumerate(sentence):
        for nb_word in sentence[max(word_index - WINDOW_SIZE, 0) :
min(word_index + WINDOW_SIZE, len(sentence)) + 1] :
            if nb_word != word:
```

```
                data.append([word, nb_word])
print('\n Original sentence:: ', corpus_raw)
print('\n **********************Input is getting prepared
**********************')
for i, eachpair in enumerate(data):
    print(i, ':', eachpair)
```

```
data
```

```
# function to convert numbers to one hot vectors
def to_one_hot(data_point_index, vocab_size):
    temp = np.zeros(vocab_size)
    temp[data_point_index] = 1
    return temp

x_train = [] # input word
y_train = [] # output word
labelss = []
for i, data_word in enumerate(data):

    x_train.append((word2int[data_word[0]]))
    if data_word[0] not in labelss:
        labelss.append(data_word[0] )
    y_train.append((word2int[ data_word[1] ]))

# convert them to numpy arrays
x_train = np.asarray(x_train)
y_train = np.asarray(y_train)
print('words: ', words)
print('X : \n', x_train)
print('\nY: \n', y_train)
```

```
labelss

y_train = y_train.reshape((-1, 1))

x_train

len(y_train)
```

```
def get_batch(size):
    assert size<len(data)
    X=[]
    Y=[]
    rdm = np.random.choice(range(len(data)), size, replace=False)

    for r in rdm:
        X.append(word2int[data[r][0]])
        Y.append([word2int[data[r][1]]])
    return X, Y

print('Batches (x, y)', get_batch(3))

get_batch(1)

get_batch(2)

get_batch(4)

BATCH_SIZE = 10
VOCAB_SIZE = vocab_size #12
EMBED_SIZE = 5
NUM_SAMPLED= 6
LEARNING_RATE =1.0 # 1e-1
X = tf.placeholder(tf.int32, shape=[BATCH_SIZE])
Y = tf.placeholder(tf.int32, shape=[BATCH_SIZE, 1])

with tf.device("/cpu:0"):
    embed_matrix = tf.Variable(tf.random_uniform([VOCAB_SIZE,
EMBED_SIZE], -1.0, 1.0)) #12,5
    embed = tf.nn.embedding_lookup(embed_matrix, X) #50 , 3

#X.shape, Y.shape, embed_matrix.shape, embed.shape

nce_weight = tf.Variable(tf.random_uniform([VOCAB_SIZE, EMBED_SIZE],-
1.0,1.0)) # (12, 5)
nce_bias = tf.Variable(tf.zeros([VOCAB_SIZE]))#12


#nce_weight, nce_bias

loss = tf.reduce_mean(tf.nn.nce_loss(weights=nce_weight,
                                     biases=nce_bias,
                                     labels=Y,
                                     inputs=embed,
                                     num_sampled=NUM_SAMPLED,
                                     num_classes=VOCAB_SIZE
                                    ))
#print (loss)
```

```
optimizer = tf.train.AdamOptimizer(1e-1).minimize(loss)

epochs = 10000
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())

    for epoch in range (epochs):
        batch_inputs, batch_labels = get_batch(BATCH_SIZE)
        _,loss_val = sess.run([optimizer,loss],feed_dict={X:
batch_inputs, Y: batch_labels})

        if epoch % 1000 == 0:
            print("Loss at", epoch, loss_val )

    temp = embed_matrix.eval()
# words


temp

len(y_train)
temp.shape

# fit a 2d PCA model to the vectors
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
trained_embeedings = pca.fit_transform(temp)
trained_embeedings

labelss
import matplotlib.pyplot as plt
#show word2vec if dim is 2
if trained_embeedings.shape[1] == 2:
    #labels = data[:10] #Show top 10 words
#     plt.xlim(-2.5, 2.4)
#     plt.ylim(-2.0, 2.2)
    for i, label in enumerate(labelss):
        x,y = trained_embeedings[i,:]
        plt.scatter(x,y)
        plt.annotate(label, xy=(x,y), xytext=(9,3),textcoords='offset
points', ha='right', va='bottom')
        #plt.savefig('word2vev.png')
    plt.show()
```

```
Vocab size:  19
Words: {'public', 'as', 'the', 'be', 'possible', 'immunization',
'accepted', 'bench', 'data', 'clinical', 'stipulated', 'also', 'that',
'policy', 'soon', 'but', 'made', 'study', 'child'}
Total number of sentenses are: 2

 Original sentence::  the bench also accepted the child immunization
policy but stipulated that the clinical study data be made public as
soon as possible.

 **********Input is getting prepared
**********
0 : ['the', 'bench']
1 : ['the', 'also']
2 : ['bench', 'the']
3 : ['bench', 'also']
4 : ['bench', 'accepted']
5 : ['also', 'the']
6 : ['also', 'bench']
7 : ['also', 'accepted']
8 : ['also', 'the']
9 : ['accepted', 'bench']
10 : ['accepted', 'also']
11 : ['accepted', 'the']
12 : ['accepted', 'child']
13 : ['the', 'also']
14 : ['the', 'accepted']
15 : ['the', 'child']
16 : ['the', 'immunization']
17 : ['child', 'accepted']
18 : ['child', 'the']
19 : ['child', 'immunization']
20 : ['child', 'policy']
21 : ['immunization', 'the']
22 : ['immunization', 'child']
23 : ['immunization', 'policy']
24 : ['immunization', 'but']
25 : ['policy', 'child']
26 : ['policy', 'immunization']
27 : ['policy', 'but']
28 : ['policy', 'stipulated']
29 : ['but', 'immunization']
30 : ['but', 'policy']
31 : ['but', 'stipulated']
32 : ['but', 'that']
33 : ['stipulated', 'policy']
34 : ['stipulated', 'but']
35 : ['stipulated', 'that']
36 : ['stipulated', 'the']
37 : ['that', 'but']
```

```
38 : ['that', 'stipulated']
39 : ['that', 'the']
40 : ['that', 'clinical']
41 : ['the', 'stipulated']
42 : ['the', 'that']
43 : ['the', 'clinical']
44 : ['the', 'study']
45 : ['clinical', 'that']
46 : ['clinical', 'the']
47 : ['clinical', 'study']
48 : ['clinical', 'data']
49 : ['study', 'the']
50 : ['study', 'clinical']
51 : ['study', 'data']
52 : ['study', 'be']
53 : ['data', 'clinical']
54 : ['data', 'study']
55 : ['data', 'be']
56 : ['data', 'made']
57 : ['be', 'study']
58 : ['be', 'data']
59 : ['be', 'made']
60 : ['be', 'public']
61 : ['made', 'data']
62 : ['made', 'be']
63 : ['made', 'public']
64 : ['made', 'as']
65 : ['public', 'be']
66 : ['public', 'made']
67 : ['public', 'as']
68 : ['public', 'soon']
69 : ['as', 'made']
70 : ['as', 'public']
71 : ['as', 'soon']
72 : ['soon', 'public']
73 : ['soon', 'as']
74 : ['soon', 'as']
75 : ['soon', 'possible']
76 : ['as', 'soon']
77 : ['as', 'possible']
78 : ['possible', 'soon']
79 : ['possible', 'as']
words:  {'public', 'as', 'the', 'be', 'possible', 'immunization',
'accepted', 'bench', 'data', 'clinical', 'stipulated', 'also', 'that',
'policy', 'soon', 'but', 'made', 'study', 'child'}
X :
 [ 2  2  7  7  7 11 11 11 11  6  6  6  6  2  2  2  2 18 18 18 18  5  5
5
  5 13 13 13 13 15 15 15 15 10 10 10 10 12 12 12 12  2  2  2  2  9  9
9
```

```
  9 17 17 17 17  8  8  8  8  3  3  3  3 16 16 16 16  0  0  0  0  1  1
1
 14 14 14 14  1  1  4  4]

Y:
 [ 7 11  2 11  6  2  7  6  2  7 11  2 18 11  6 18  5  6  2  5 13  2 18
13
 15 18  5 15 10  5 13 10 12 13 15 12  2 15 10  2  9 10 12  9 17 12  2
17
  8  2  9  8  3  9 17  3 16 17  8 16  0  8  3  0  1  3 16  1 14 16  0
14
  0  1  1  4 14  4 14  1]
Batches (x, y) ([10, 6, 5], [[2], [7], [2]])
Loss at 0 7.4335556
Loss at 1000 1.8706411
Loss at 2000 1.9462725
Loss at 3000 1.6148535
Loss at 4000 2.1192715
Loss at 5000 2.1654727
Loss at 6000 1.5511638
Loss at 7000 1.7983822
Loss at 8000 2.226023
Loss at 9000 1.713986
```