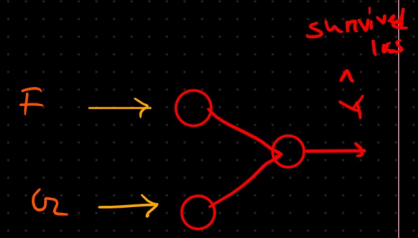
Agenda:

1. RNN – Recurrent Neural Networks
2. What is Sequential Data/ Learning?
3. Some Drawbacks in ANN and CNN
4. Types of RNN Networks
5. Practical Demo of RNN using Keras (Sentiment Analysis)

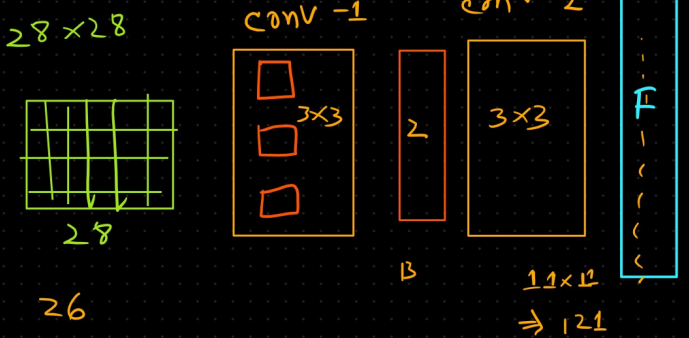
ANN → Tabular Data

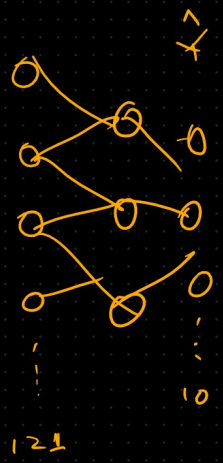
Gr	Fare	Survived
F	20K	1
M	5K	0
-	-	-
-	-	-

X Y

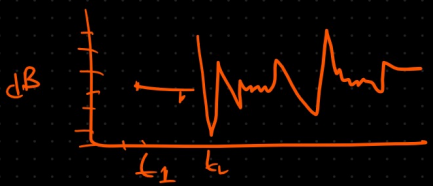


CNN → Image 28





1. My name is Buppy
   t_1 t_2 t_3 t_4
2. I love AI



→ Text
→ Time Series
→ Speech
→ DNA seq

* Why should use RNN?

5	Hi my name is Bappy	0
3	I love AI	1
4	Bappy is My name	0

Total : 12

Hi ⇒ [1, 0, 0, 0, ---- 12]

My ⇒ [0, 1, 0, 0, ---- 12]

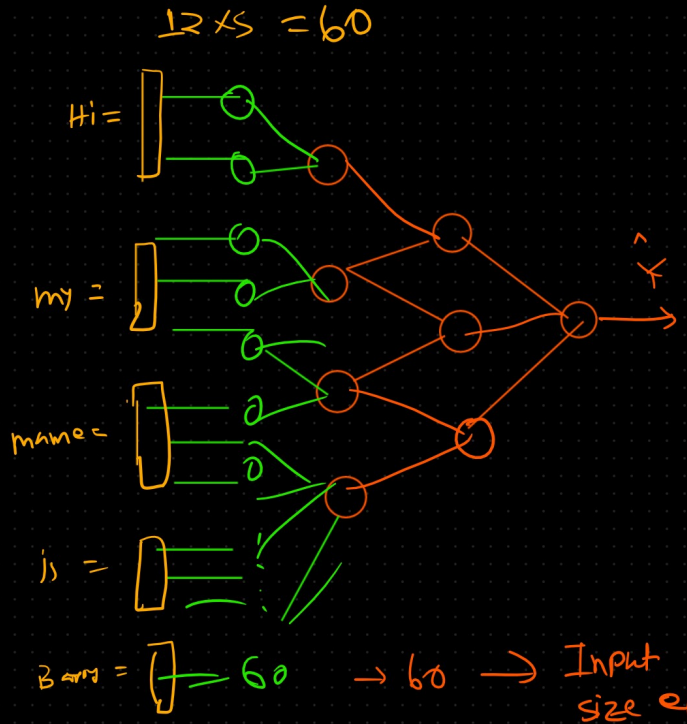
---

(2) [1, 0 0 0 .. 12)

[0, 1, 0 ---]

[0 0, 1 .. 12]

12 × 5 = 60



→ 60 → Input size error!

12 × 3 ⇒ 36

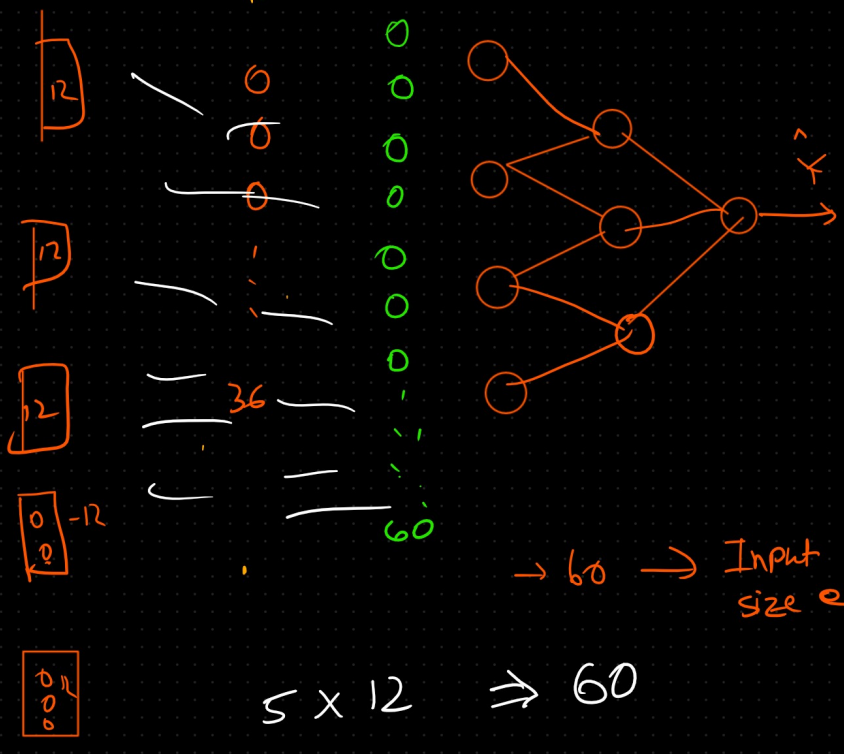
12 × 4 ⇒ 48

Hand coded Solution:

28 × 28

32 × 2

32 × 32



→ 60 → Input size error!

5 × 12 ⇒ 60

* Note all problems:

① Text input → varying size

② zero padding → sparse Matrix → Unecssary computation

③ loosing Sequential Info (context meaning)

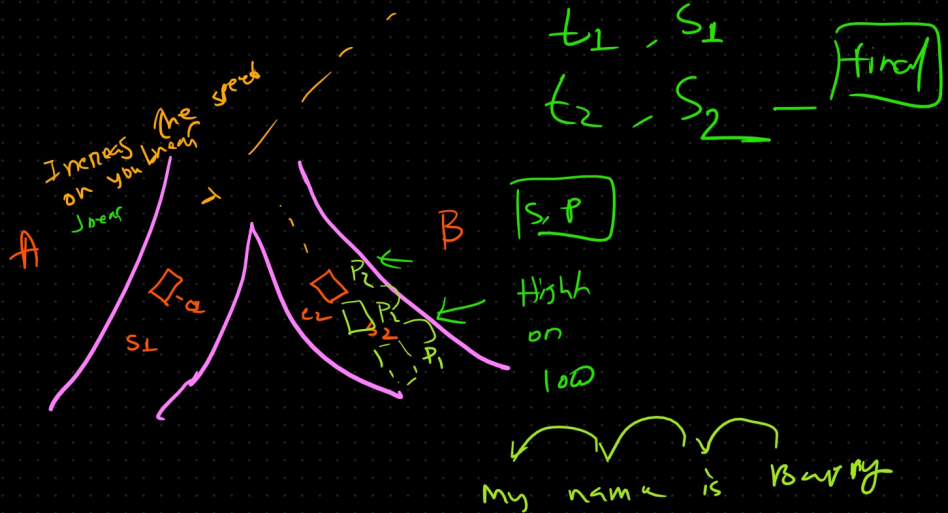
RNN – Recurrent Neural Network

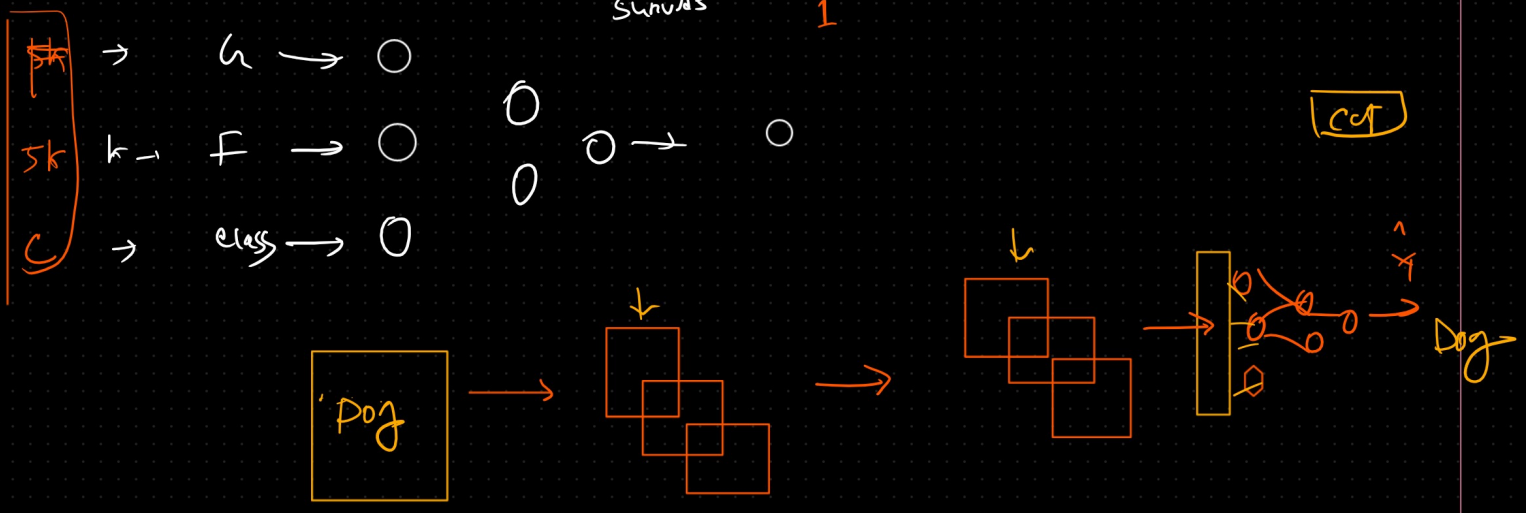
→ 1980

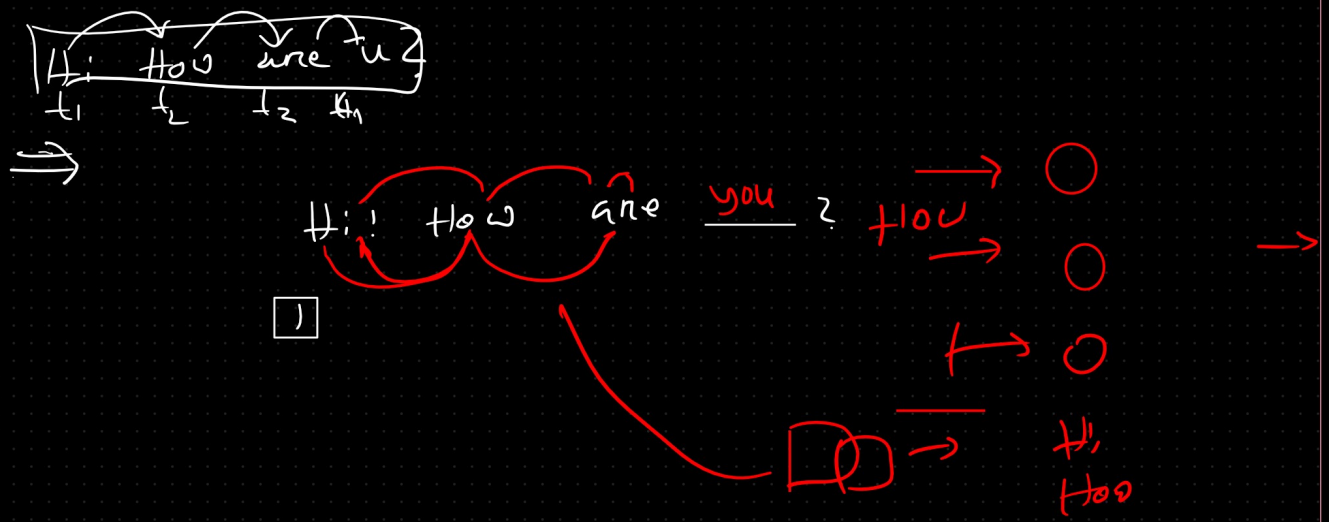
Demo:

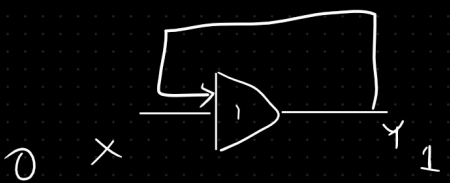
Self driven →

RL →









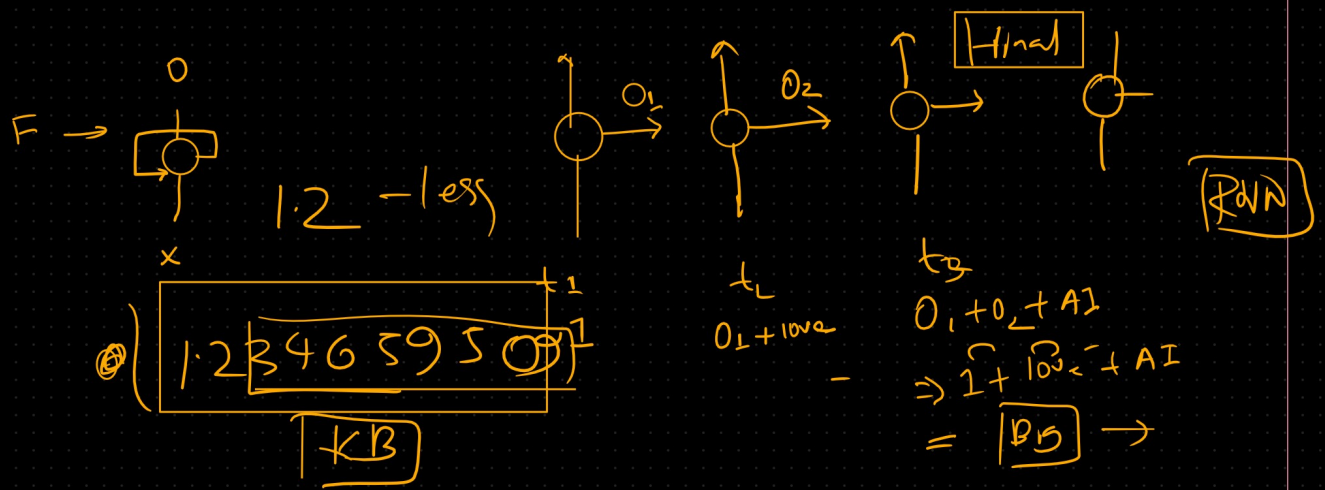
RNN → Recurrent Neural Network

Recursion

DSA

[I love AI]

(My name is Kumar)



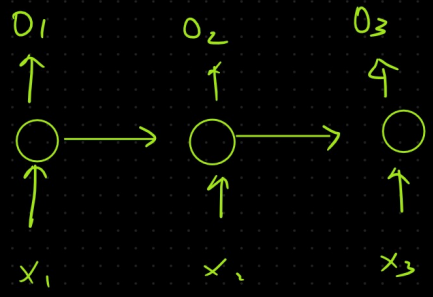
1.20345657 → Right KB

1.2 → less → KB

RNN → ① LSTM ② GRU → memory cell

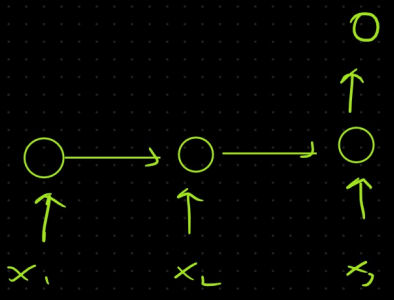
Types of RNN / Sequential Architecture:

Many to many: (seq to seq)



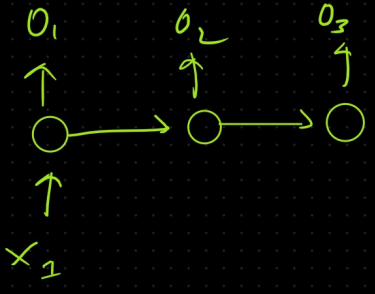
⇒ Translator

Many to one (seq to vec)



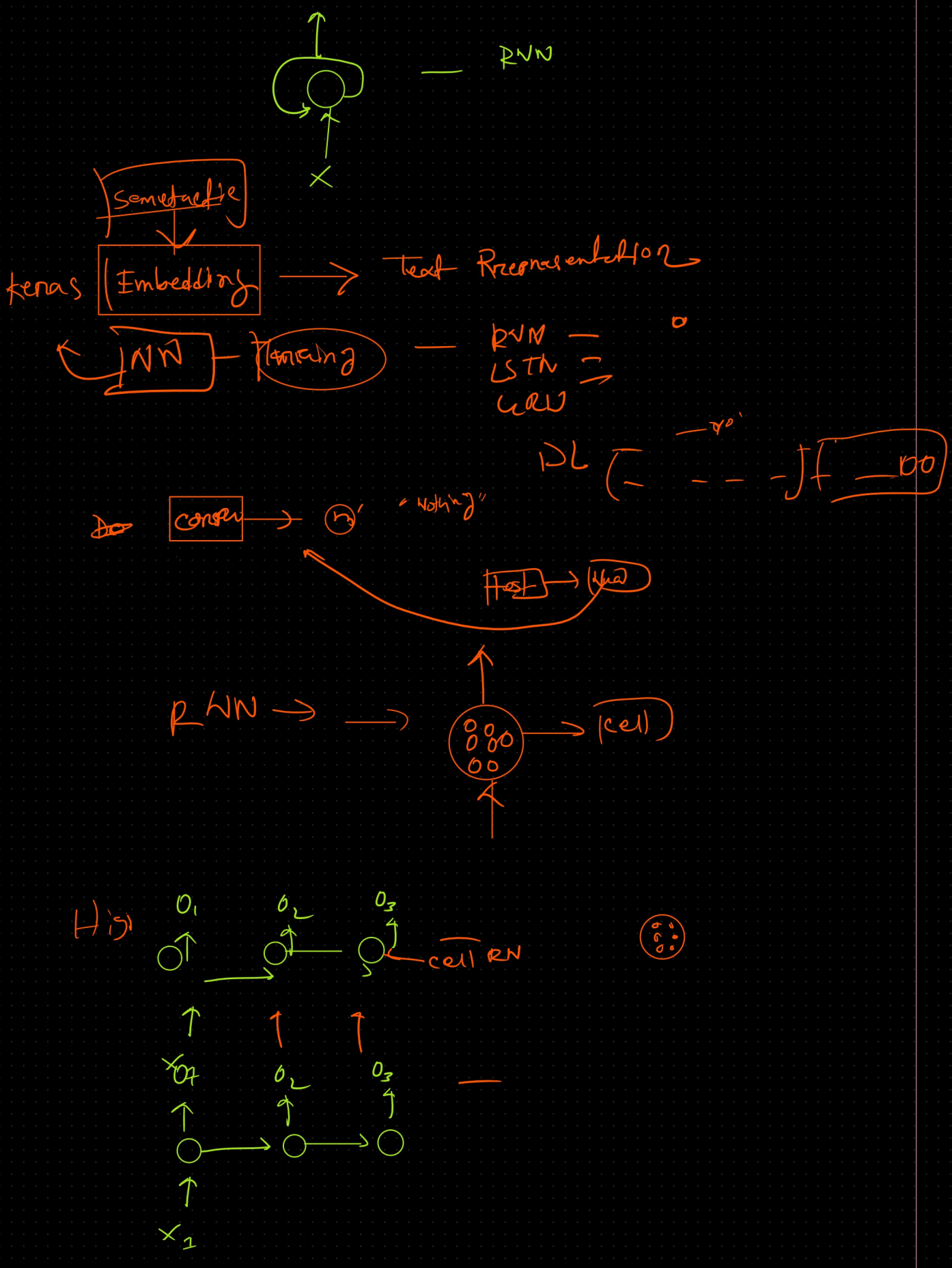
⇒ Text classification

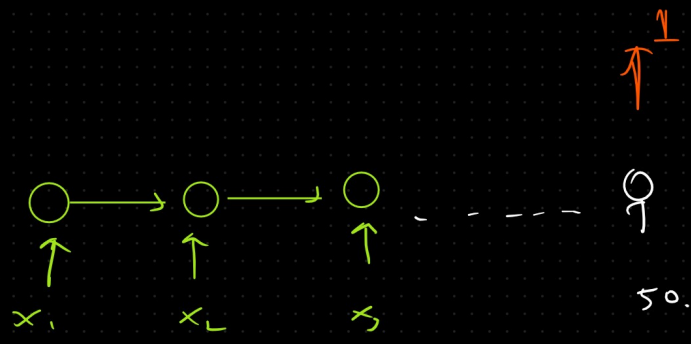
One to many (vec to seq):

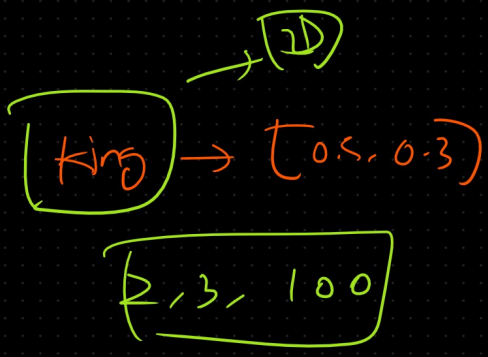


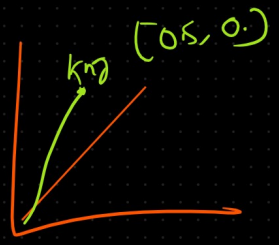
⇒ Text generation

One to one (vec to vec):




1
x_1
x_2
x_3
50.


king
[0.5, 0.3)
2, 3, 100


king
[0.5, 0.)

IMDB data: https://www.kaggle.com/datasets/lakshmi25npathi/imdb-dataset-of-50k-movie-reviews (https://www.kaggle.com/datasets/lakshmi25npathi/imdb-dataset-of-50k-movie-reviews)

How to do integer encoding using keras

In [1]:
```
import numpy as np
```

In [2]:
```
docs = ['recurrent neural network',
        'neural network',
        'artificial neural',
        'connections between nodes',
        'can create a cycle',
        'allowing output',
        'some nodes to affect subsequent',
        'exhibit temporal',
        'dynamic behavior',
        'type of Neural Network',
    'affect subsequent']
```

In [3]:
```
from keras.preprocessing.text import Tokenizer
tokenizer = Tokenizer(oov_token='<nothing>')
```

In [4]:
```
tokenizer.fit_on_texts(docs)
```

In [5]:
```
tokenizer.word_index
```

Out[5]:
```
{'<nothing>': 1,
 'neural': 2,
 'network': 3,
 'nodes': 4,
 'affect': 5,
 'subsequent': 6,
 'recurrent': 7,
 'artificial': 8,
 'connections': 9,
 'between': 10,
 'can': 11,
 'create': 12,
 'a': 13,
 'cycle': 14,
 'allowing': 15,
 'output': 16,
 'some': 17,
 'to': 18,
 'exhibit': 19,
 'temporal': 20,
 'dynamic': 21,
 'behavior': 22,
 'type': 23,
 'of': 24}
```

In [6]:
```
tokenizer.word_counts
```

Out[6]:
```
OrderedDict([('recurrent', 1),
             ('neural', 4),
             ('network', 3),
             ('artificial', 1),
             ('connections', 1),
             ('between', 1),
             ('nodes', 2),
             ('can', 1),
             ('create', 1),
             ('a', 1),
             ('cycle', 1),
             ('allowing', 1),
             ('output', 1),
             ('some', 1),
             ('to', 1),
             ('affect', 2),
             ('subsequent', 2),
             ('exhibit', 1),
             ('temporal', 1),
             ('dynamic', 1),
             ('behavior', 1),
             ('type', 1),
             ('of', 1)])
```

In [7]:
```
tokenizer.document_count
```

Out[7]:
```
11
```

In [8]:
```
sequences = tokenizer.texts_to_sequences(docs)
sequences
```

Out[8]:
```
[[7, 2, 3],
 [2, 3],
 [8, 2],
 [9, 10, 4],
 [11, 12, 13, 14],
 [15, 16],
 [17, 4, 18, 5, 6],
 [19, 20],
 [21, 22],
 [23, 24, 2, 3],
 [5, 6]]
```

In [9]:
```
from keras.utils import pad_sequences
```

In [10]:
```
sequences = pad_sequences(sequences,padding='post')
```

In [11]:
```
sequences
```

Out[11]:
```
array([[ 7,  2,  3,  0,  0],
       [ 2,  3,  0,  0,  0],
       [ 8,  2,  0,  0,  0],
       [ 9, 10,  4,  0,  0],
       [11, 12, 13, 14,  0],
       [15, 16,  0,  0,  0],
       [17,  4, 18,  5,  6],
       [19, 20,  0,  0,  0],
       [21, 22,  0,  0,  0],
       [23, 24,  2,  3,  0],
       [ 5,  6,  0,  0,  0]], dtype=int32)
```

Sentiment Analysis

In [12]:
```
from keras.datasets import imdb
from keras import Sequential
from keras.layers import Dense,SimpleRNN,Embedding,Flatten
```

In [13]:
```
(X_train,y_train),(X_test,y_test) = imdb.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb.
npz (https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb.npz)
17464789/17464789 [==============================] - 2s 0us/step
```

In [16]:
```
y_test
```

Out[16]:
```
array([0, 1, 1, ..., 0, 0, 0])
```

In [15]:
```
X_train
```

Out[15]:
```
array([list([1, 14, 22, 16, 43, 530, 973, 1622, 1385, 65, 458, 4468, 66, 3941, 4, 173,
36, 256, 5, 25, 100, 43, 838, 112, 50, 670, 22665, 9, 35, 480, 284, 5, 150, 4, 172, 11
2, 167, 21631, 336, 385, 39, 4, 172, 4536, 1111, 17, 546, 38, 13, 447, 4, 192, 50, 16,
6, 147, 2025, 19, 14, 22, 4, 1920, 4613, 469, 4, 22, 71, 87, 12, 16, 43, 530, 38, 76, 1
5, 13, 1247, 4, 22, 17, 515, 17, 12, 16, 626, 18, 19193, 5, 62, 386, 12, 8, 316, 8, 10
6, 5, 4, 2223, 5244, 16, 480, 66, 3785, 33, 4, 130, 12, 16, 38, 619, 5, 25, 124, 51, 3
6, 135, 48, 25, 1415, 33, 6, 22, 12, 215, 28, 77, 52, 5, 14, 407, 16, 82, 10311, 8, 4,
107, 117, 5952, 15, 256, 4, 31050, 7, 3766, 5, 723, 36, 71, 43, 530, 476, 26, 400, 317,
46, 7, 4, 12118, 1029, 13, 104, 88, 4, 381, 15, 297, 98, 32, 2071, 56, 26, 141, 6, 194,
7486, 18, 4, 226, 22, 21, 134, 476, 26, 480, 5, 144, 30, 5535, 18, 51, 36, 28, 224, 92,
25, 104, 4, 226, 65, 16, 38, 1334, 88, 12, 16, 283, 5, 16, 4472, 113, 103, 32, 15, 16,
5345, 19, 178, 32]),
       list([1, 194, 1153, 194, 8255, 78, 228, 5, 6, 1463, 4369, 5012, 134, 26, 4, 715,
8, 118, 1634, 14, 394, 20, 13, 119, 954, 189, 102, 5, 207, 110, 3103, 21, 14, 69, 188,
8, 30, 23, 7, 4, 249, 126, 93, 4, 114, 9, 2300, 1523, 5, 647, 4, 116, 9, 35, 8163, 4, 2
29, 9, 340, 1322, 4, 118, 9, 4, 130, 4901, 19, 4, 1002, 5, 89, 29, 952, 46, 37, 4, 455,
9, 45, 43, 38, 1543, 1905, 398, 4, 1649, 26, 6853, 5, 163, 11, 3215, 10156, 4, 1153, 9,
194, 775, 7, 8255, 11596, 349, 2637, 148, 605, 15358, 8003, 15, 123, 125, 68, 23141, 68
53, 15, 349, 165, 4362, 98, 5, 4, 228, 9, 43, 36893, 1157, 15, 299, 120, 5, 120, 174, 1
1, 220, 175, 136, 50, 9, 4373, 228, 8255, 5, 25249, 656, 245, 2350, 5, 4, 9837, 131, 15
2, 491, 18, 46151, 32, 7464, 1212, 14, 9, 6, 371, 78, 22, 625, 64, 1382, 9, 8, 168, 14
5, 23, 4, 1690, 15, 16, 4, 1355, 5, 28, 6, 52, 154, 462, 33, 89, 78, 285, 16, 145, 9
5]),
       list([1, 14, 47, 8, 30, 31, 7, 4, 249, 108, 7, 4, 5974, 54, 61, 369, 13, 71, 14
9, 14, 22, 112, 4, 2401, 311, 12, 16, 3711, 33, 75, 43, 1829, 296, 4, 86, 320, 35, 534,
19, 263, 4821, 1301, 4, 1873, 33, 89, 78, 12, 66, 16, 4, 360, 7, 4, 58, 316, 334, 11,
4, 1716, 43, 645, 662, 8, 257, 85, 1200, 42, 1228, 2578, 83, 68, 3912, 15, 36, 165, 153
9, 278, 36, 69, 44076, 780, 8, 106, 14, 6905, 1338, 18, 6, 22, 12, 215, 28, 610, 40, 6,
87, 326, 23, 2300, 21, 23, 22, 12, 272, 40, 57, 31, 11, 4, 22, 47, 6, 2307, 51, 9, 170,
23, 595, 116, 595, 1352, 13, 191, 79, 638, 89, 51428, 14, 9, 8, 106, 607, 624, 35, 534,
6, 227, 7, 129, 113]),
       ...,
       list([1, 11, 6, 230, 245, 6401, 9, 6, 1225, 446, 86527, 45, 2174, 84, 8322, 400
7, 21, 4, 912, 84, 14532, 325, 725, 134, 15271, 1715, 84, 5, 36, 28, 57, 1099, 21, 8, 1
40, 8, 703, 5, 11656, 84, 56, 18, 1644, 14, 9, 31, 7, 4, 9406, 1209, 2295, 26094, 1008,
18, 6, 20, 207, 110, 563, 12, 8, 2901, 17793, 8, 97, 6, 20, 53, 4767, 74, 4, 460, 364,
1273, 29, 270, 11, 960, 108, 45, 40, 29, 2961, 395, 11, 6, 4065, 500, 7, 14492, 89, 36
4, 70, 29, 140, 4, 64, 4780, 11, 4, 2678, 26, 178, 4, 529, 443, 17793, 5, 27, 710, 117,
74936, 8123, 165, 47, 84, 37, 131, 818, 14, 595, 10, 10, 61, 1242, 1209, 10, 10, 288, 2
260, 1702, 34, 2901, 17793, 4, 65, 496, 4, 231, 7, 790, 5, 6, 320, 234, 2766, 234, 111
9, 1574, 7, 496, 4, 139, 929, 2901, 17793, 7750, 5, 4241, 18, 4, 8497, 13164, 250, 11,
1818, 7561, 4, 4217, 5408, 747, 1115, 372, 1890, 1006, 541, 9303, 7, 4, 59, 11027, 4, 3
586, 22459]),
       list([1, 1446, 7079, 69, 72, 3305, 13, 610, 930, 8, 12, 582, 23, 5, 16, 484, 68
5, 54, 349, 11, 4120, 2959, 45, 58, 1466, 13, 197, 12, 16, 43, 23, 21469, 5, 62, 30, 14
5, 402, 11, 4131, 51, 575, 32, 61, 369, 71, 66, 770, 12, 1054, 75, 100, 2198, 8, 4, 10
5, 37, 69, 147, 712, 75, 3543, 44, 257, 390, 5, 69, 263, 514, 105, 50, 286, 1814, 23,
4, 123, 13, 161, 40, 5, 421, 4, 116, 16, 897, 13, 40691, 40, 319, 5872, 112, 6700, 11,
4803, 121, 25, 70, 3468, 4, 719, 3798, 13, 18, 31, 62, 40, 8, 7200, 4, 29455, 7, 14, 12
3, 5, 942, 25, 8, 721, 12, 145, 5, 202, 12, 160, 580, 202, 12, 6, 52, 58, 11418, 92, 40
1, 728, 12, 39, 14, 251, 8, 15, 251, 5, 21213, 12, 38, 84, 80, 124, 12, 9, 23]),
       list([1, 17, 6, 194, 337, 7, 4, 204, 22, 45, 254, 8, 106, 14, 123, 4, 12815, 27
0, 14437, 5, 16923, 12255, 732, 2098, 101, 405, 39, 14, 1034, 4, 1310, 9, 115, 50, 305,
12, 47, 4, 168, 5, 235, 7, 38, 111, 699, 102, 7, 4, 4039, 9245, 9, 24, 6, 78, 1099, 17,
2345, 16553, 21, 27, 9685, 6139, 5, 29043, 1603, 92, 1183, 4, 1310, 7, 4, 204, 42, 97,
90, 35, 221, 109, 29, 127, 27, 118, 8, 97, 12, 157, 21, 6789, 85010, 9, 6, 66, 78, 109
9, 4, 631, 1191, 5, 2642, 272, 191, 1070, 6, 7585, 8, 2197, 70907, 10755, 544, 5, 383,
1271, 848, 1468, 12183, 497, 16876, 8, 1597, 8778, 19280, 21, 60, 27, 239, 9, 43, 8368,
209, 405, 10, 10, 12, 764, 40, 4, 248, 20, 12, 16, 5, 174, 1791, 72, 7, 51, 6, 1739, 2
2, 4, 204, 131, 9])],
      dtype=object)
```

In [17]:
```
print(len(X_train[2]))
print(len(X_train[3]))
```

```
141
550
```

In [18]:
```
X_train = pad_sequences(X_train,padding='post',maxlen=50)
X_test = pad_sequences(X_test,padding='post',maxlen=50)
```

In [19]:
```
X_train[0]
```

Out[19]:
```
array([2071,   56,   26,  141,    6,  194, 7486,   18,    4,  226,   22,
         21,  134,  476,   26,  480,    5,  144,   30, 5535,   18,   51,
         36,   28,  224,   92,   25,  104,    4,  226,   65,   16,   38,
       1334,   88,   12,   16,  283,    5,   16, 4472,  113,  103,   32,
         15,   16, 5345,   19,  178,   32], dtype=int32)
```

In [20]:
```
print(len(X_train[2]))
print(len(X_train[3]))
```

```
50
50
```

In [21]:
```
model = Sequential()

model.add(SimpleRNN(32,input_shape=(50,1),return_sequences=False))
model.add(Dense(1,activation='sigmoid'))

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 simple_rnn (SimpleRNN)      (None, 32)                1088

 dense (Dense)               (None, 1)                 33

=================================================================
Total params: 1121 (4.38 KB)
Trainable params: 1121 (4.38 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [22]:
```
model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])

model.fit(X_train,y_train,epochs=5,validation_data=(X_test,y_test))
```

```
Epoch 1/5
782/782 [==============================] - 38s 42ms/step - loss: 0.6976 - accuracy: 0.5
046 - val_loss: 0.6952 - val_accuracy: 0.4992
Epoch 2/5
782/782 [==============================] - 32s 42ms/step - loss: 0.6935 - accuracy: 0.5
013 - val_loss: 0.6947 - val_accuracy: 0.5026
Epoch 3/5
782/782 [==============================] - 32s 42ms/step - loss: 0.6932 - accuracy: 0.5
054 - val_loss: 0.6938 - val_accuracy: 0.5039
Epoch 4/5
782/782 [==============================] - 32s 41ms/step - loss: 0.6931 - accuracy: 0.5
059 - val_loss: 0.6946 - val_accuracy: 0.5008
Epoch 5/5
782/782 [==============================] - 33s 42ms/step - loss: 0.6927 - accuracy: 0.5
101 - val_loss: 0.6945 - val_accuracy: 0.5040
```

Out[22]:
```
<keras.src.callbacks.History at 0x7a82b3c49000>
```

How to do encodings using keras embeddings

In [23]:
```
docs = ['recurrent neural network',
        'neural network',
        'artificial neural',
        'connections between nodes',
        'can create a cycle',
        'allowing output',
        'some nodes to affect subsequent',
        'exhibit temporal',
        'dynamic behavior',
        'type of Neural Network',
    'affect subsequent']
```

In [24]:
```
from keras.preprocessing.text import Tokenizer
tokenizer = Tokenizer()
```

In [25]:
```
tokenizer.fit_on_texts(docs)
```

In [26]:
```
len(tokenizer.word_index)
```

Out[26]:
```
23
```

In [27]:
```
sequences = tokenizer.texts_to_sequences(docs)
sequences
```

Out[27]:
```
[[6, 1, 2],
 [1, 2],
 [7, 1],
 [8, 9, 3],
 [10, 11, 12, 13],
 [14, 15],
 [16, 3, 17, 4, 5],
 [18, 19],
 [20, 21],
 [22, 23, 1, 2],
 [4, 5]]
```

In [28]:
```
from keras.utils import pad_sequences
sequences = pad_sequences(sequences,padding='post')
sequences
```

Out[28]:
```
array([[ 6,  1,  2,  0,  0],
       [ 1,  2,  0,  0,  0],
       [ 7,  1,  0,  0,  0],
       [ 8,  9,  3,  0,  0],
       [10, 11, 12, 13,  0],
       [14, 15,  0,  0,  0],
       [16,  3, 17,  4,  5],
       [18, 19,  0,  0,  0],
       [20, 21,  0,  0,  0],
       [22, 23,  1,  2,  0],
       [ 4,  5,  0,  0,  0]], dtype=int32)
```

In [30]:
```
model = Sequential()
model.add(Embedding(23,output_dim=2,input_length=5)) #Total vocab len, ouput dim(per wor

model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_1 (Embedding)     (None, 5, 2)              46

=================================================================
Total params: 46 (184.00 Byte)
Trainable params: 46 (184.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [31]:
```
model.compile('adam','accuracy')
```

In [32]:
```
pred = model.predict(sequences)
print(pred)
```

```
1/1 [==============================] - 0s 70ms/step
[[[ 0.02842501 -0.03834099]
  [ 0.02416212  0.01598806]
  [-0.04352813  0.03636582]
  [-0.03299248  0.03105232]
  [-0.03299248  0.03105232]]

 [[ 0.02416212  0.01598806]
  [-0.04352813  0.03636582]
  [-0.03299248  0.03105232]
  [-0.03299248  0.03105232]
  [-0.03299248  0.03105232]]

 [[-0.04227873  0.02836833]
  [ 0.02416212  0.01598806]
  [-0.03299248  0.03105232]
  [-0.03299248  0.03105232]
  [-0.03299248  0.03105232]]

 [[ 0.02807537  0.02820093]
  [-0.02432675  0.0137653 ]
  [ 0.03763125 -0.04061527]
  [-0.03299248  0.03105232]
  [-0.03299248  0.03105232]]

 [[-0.04817903 -0.00214241]
  [ 0.03807466  0.00347348]
  [ 0.00501565 -0.03119435]
  [-0.02733684  0.01278828]
  [-0.03299248  0.03105232]]

 [[-0.00200746  0.01143809]
  [ 0.00215219 -0.00452148]
  [-0.03299248  0.03105232]
  [-0.03299248  0.03105232]
  [-0.03299248  0.03105232]]

 [[-0.01166817  0.020961  ]
  [ 0.03763125 -0.04061527]
  [ 0.03656984 -0.03942559]
  [ 0.01513976  0.04213302]
  [ 0.00669478  0.00938103]]

 [[ 0.01795432 -0.00754835]
  [ 0.04522084 -0.04998295]
  [-0.03299248  0.03105232]
  [-0.03299248  0.03105232]
  [-0.03299248  0.03105232]]

 [[-0.04471531  0.03016635]
  [-0.04119124 -0.0185082 ]
  [-0.03299248  0.03105232]
  [-0.03299248  0.03105232]
  [-0.03299248  0.03105232]]

 [[ 0.01991191  0.02911988]
  [ 0.          0.        ]
  [ 0.02416212  0.01598806]
  [-0.04352813  0.03636582]
  [-0.03299248  0.03105232]]

 [[ 0.01513976  0.04213302]
  [ 0.00669478  0.00938103]
  [-0.03299248  0.03105232]
  [-0.03299248  0.03105232]
  [-0.03299248  0.03105232]]]
```

In [33]:
```
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
```

In [34]:
```
(X_train,y_train),(X_test,y_test) = imdb.load_data()
```

In [35]:
```
X_train = pad_sequences(X_train,padding='post',maxlen=50)
X_test = pad_sequences(X_test,padding='post',maxlen=50)
```

In [36]:
```
X_train.shape
```

Out[36]:
```
(25000, 50)
```

In [37]:
```
model = Sequential()
model.add(Embedding(10000,output_dim=2,input_length=50))
model.add(SimpleRNN(32,return_sequences=False))
model.add(Dense(1, activation='sigmoid'))

model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_2 (Embedding)     (None, 50, 2)             20000

 simple_rnn_1 (SimpleRNN)    (None, 32)                1120

 dense_1 (Dense)             (None, 1)                 33

=================================================================
Total params: 21153 (82.63 KB)
Trainable params: 21153 (82.63 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [38]:
```
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['acc'])
history = model.fit(X_train, y_train,epochs=1,validation_data=(X_test,y_test))
```

```
782/782 [==============================] - 75s 93ms/step - loss: 0.6383 - acc: 0.5972 -
val_loss: 0.4480 - val_acc: 0.7931
```

Predictions

In [40]:
```
X_test[0][0:50].shape
```

Out[40]:
```
(50,)
```

In [41]:
```
X_test[0][0:50].reshape(1,-1).shape
```

Out[41]:
```
(1, 50)
```

In [42]:
```
test_data = X_test[0][0:50].reshape(1,-1)
```

In [43]:
```
model.predict(test_data)
```

```
1/1 [==============================] - 0s 129ms/step
```

Out[43]:
```
array([[0.27053547]], dtype=float32)
```

If it is 1 : mean positve and less than 1 mean Negative review