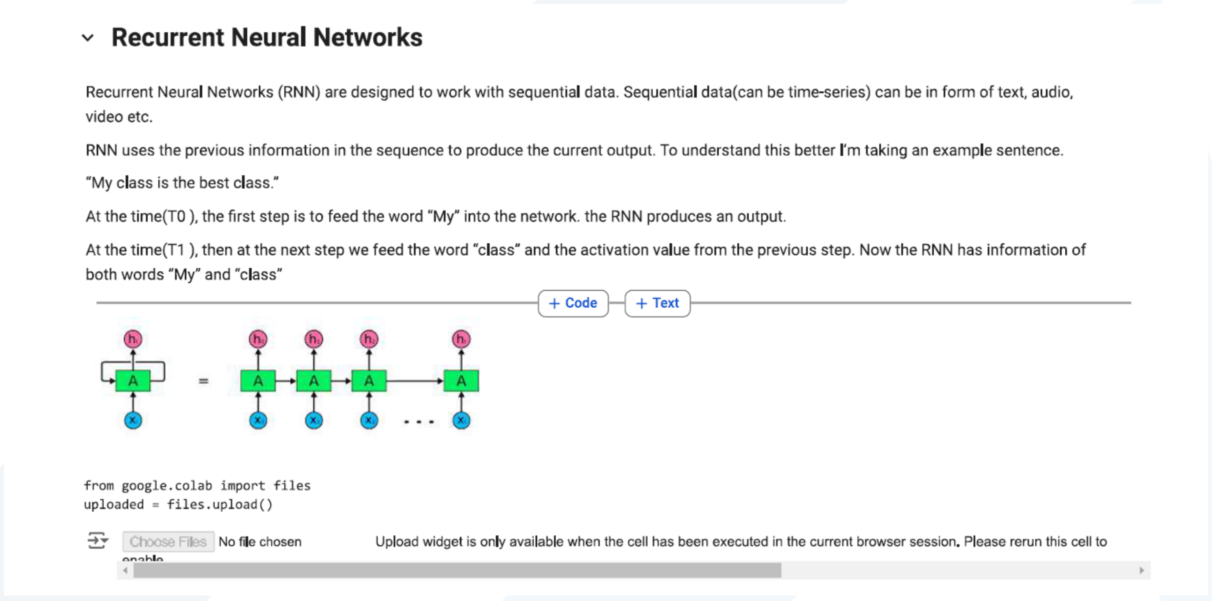
الغاية من الجلسة:

- التعامل مع Tensorflow وبناء RNN model
- التعرف على بعض الطرق للتعامل مع مشكلة over fitting



Recurrent Neural Networks

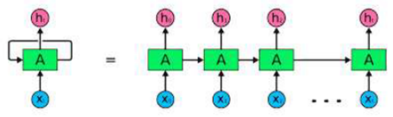
Recurrent Neural Networks (RNN) are designed to work with sequential data. Sequential data(can be time-series) can be in form of text, audio, video etc.

RNN uses the previous information in the sequence to produce the current output. To understand this better I'm taking an example sentence.

"My class is the best class."

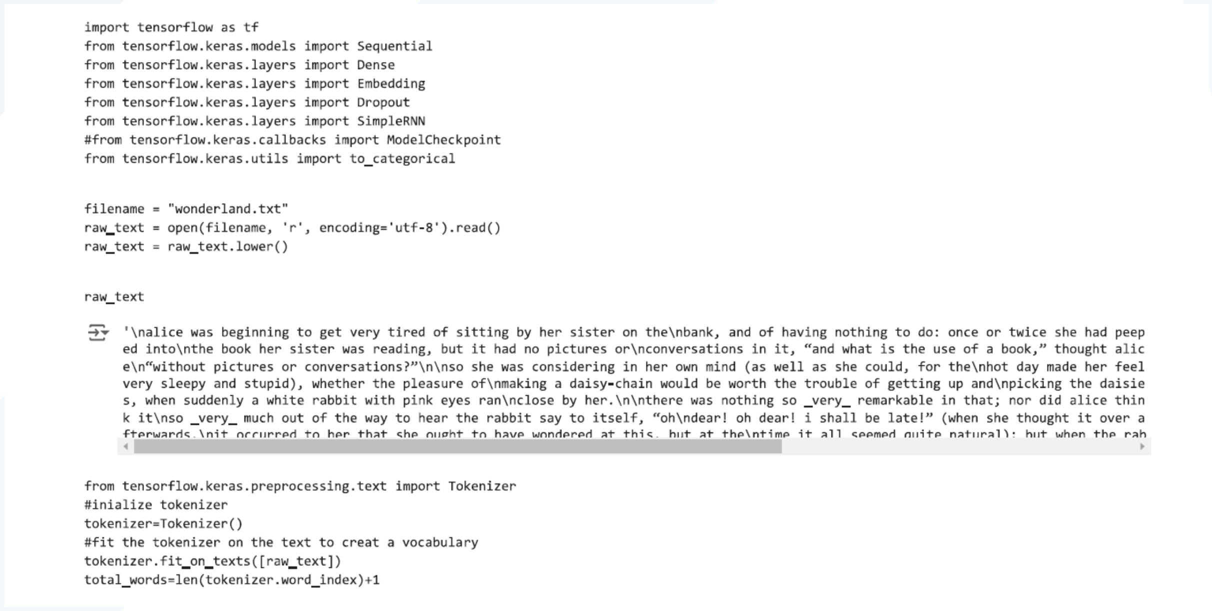
At the time(T0), the first step is to feed the word "My" into the network. the RNN produces an output.

At the time(T1), then at the next step we feed the word "class" and the activation value from the previous step. Now the RNN has information of both words "My" and "class"



```
from google.colab import files
uploaded = files.upload()
```

```
Choose Files No file chosen          Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable
```



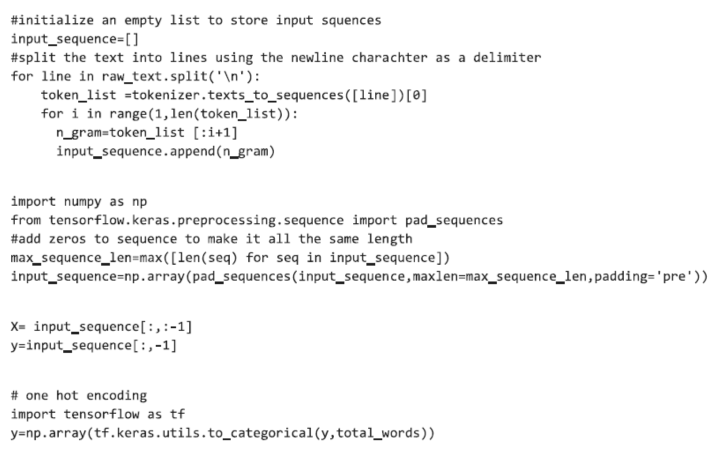
```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Embedding
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import SimpleRNN
#from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.utils import to_categorical
```

```
filename = "wonderland.txt"
raw_text = open(filename, 'r', encoding='utf-8').read()
raw_text = raw_text.lower()
```

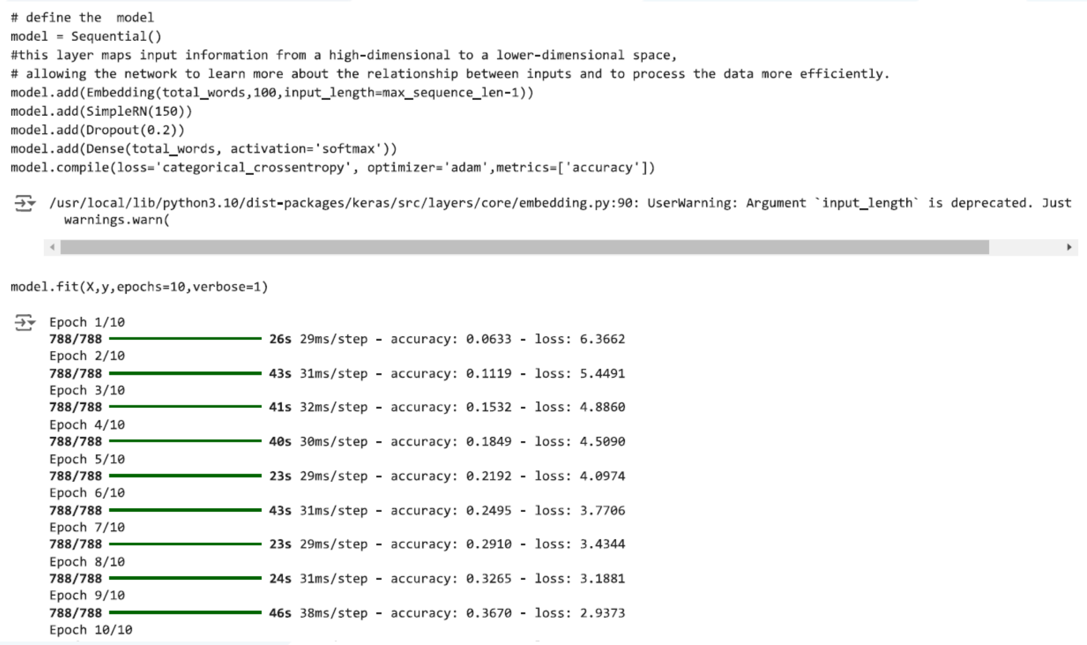
```
raw_text
```

```
'\nalice was beginning to get very tired of sitting by her sister on the\nbank, and of having nothing to do: once or twice she had peep
ed into\nthe book her sister was reading, but it had no pictures or\nconversations in it, "and what is the use of a book," thought alic
e\n"without pictures or conversations?"\n\nso she was considering in her own mind (as well as she could, for the\nhot day made her feel
very sleepy and stupid), whether the pleasure of\nmaking a daisy-chain would be worth the trouble of getting up and\npicking the daisie
s, when suddenly a white rabbit with pink eyes ran\nclose by her.\n\nthere was nothing so _very_ remarkable in that; nor did alice thin
k it\nso _very_ much out of the way to hear the rabbit say to itself, "oh\ndear! oh dear! i shall be late!" (when she thought it over a
fterwards.\nit occurred to her that she ought to have wondered at this, but at the\ntime it all seemed quite natural); but when the rab
```

```
from tensorflow.keras.preprocessing.text import Tokenizer
#inialize tokenizer
tokenizer=Tokenizer()
#fit the tokenizer on the text to creat a vocabulary
tokenizer.fit_on_texts([raw_text])
total_words=len(tokenizer.word_index)+1
```



```
#initialize an empty list to store input squences
input_sequence=[]
#split the text into lines using the newline charachter as a delimiter
for line in raw_text.split('\n'):
    token_list =tokenizer.texts_to_sequences([line])[0]
    for i in range(1,len(token_list)):
      n_gram=token_list [:i+1]
      input_sequence.append(n_gram)


import numpy as np
from tensorflow.keras.preprocessing.sequence import pad_sequences
#add zeros to sequence to make it all the same length
max_sequence_len=max([len(seq) for seq in input_sequence])
input_sequence=np.array(pad_sequences(input_sequence,maxlen=max_sequence_len,padding='pre'))


X= input_sequence[:,:-1]
y=input_sequence[:,-1]


# one hot encoding
import tensorflow as tf
y=np.array(tf.keras.utils.to_categorical(y,total_words))
```



```
# define the  model
model = Sequential()
#this layer maps input information from a high-dimensional to a lower-dimensional space,
# allowing the network to learn more about the relationship between inputs and to process the data more efficiently.
model.add(Embedding(total_words,100,input_length=max_sequence_len-1))
model.add(SimpleRN(150))
model.add(Dropout(0.2))
model.add(Dense(total_words, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam',metrics=['accuracy'])
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/embedding.py:90: UserWarning: Argument `input_length` is deprecated. Just
  warnings.warn(
```

```
model.fit(X,y,epochs=10,verbose=1)
```

```
Epoch 1/10
788/788 ━━━━━━━━━━━━━━━━━━━━ 26s 29ms/step - accuracy: 0.0633 - loss: 6.3662
Epoch 2/10
788/788 ━━━━━━━━━━━━━━━━━━━━ 43s 31ms/step - accuracy: 0.1119 - loss: 5.4491
Epoch 3/10
788/788 ━━━━━━━━━━━━━━━━━━━━ 41s 32ms/step - accuracy: 0.1532 - loss: 4.8860
Epoch 4/10
788/788 ━━━━━━━━━━━━━━━━━━━━ 40s 30ms/step - accuracy: 0.1849 - loss: 4.5090
Epoch 5/10
788/788 ━━━━━━━━━━━━━━━━━━━━ 23s 29ms/step - accuracy: 0.2192 - loss: 4.0974
Epoch 6/10
788/788 ━━━━━━━━━━━━━━━━━━━━ 43s 31ms/step - accuracy: 0.2495 - loss: 3.7706
Epoch 7/10
788/788 ━━━━━━━━━━━━━━━━━━━━ 23s 29ms/step - accuracy: 0.2910 - loss: 3.4344
Epoch 8/10
788/788 ━━━━━━━━━━━━━━━━━━━━ 24s 31ms/step - accuracy: 0.3265 - loss: 3.1881
Epoch 9/10
788/788 ━━━━━━━━━━━━━━━━━━━━ 46s 38ms/step - accuracy: 0.3670 - loss: 2.9373
Epoch 10/10
```

```
seed_text='i want to '
next_words =3
for i in range (next_words):
  token_list=tokenizer.texts_to_sequences ([seed_text])[0]
  token_list=pad_sequences([token_list],maxlen=max_sequence_len-1,padding='pre')
  predicted=np.argmax(model.predict(token_list),axis=-1)
  output_word=''
  for word,index in tokenizer.word_index.items():
    if index==predicted:
      output_word=word
      break
seed_text+=' '+output_word
print(seed_text)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
i want to be
```

Some Techniques to Prevent Overfitting in Neural Networks

1. Simplifying The Model

The first step when dealing with overfitting is to decrease the complexity of the model. To decrease the complexity, we can simply remove layers or reduce the number of neurons to make the network smaller. While doing this, it is important to calculate the input and output dimensions of the various layers involved in the neural network. There is no general rule on how much to remove or how large your network should be. But, if your neural network is overfitting, try making it smaller.

2. Early Stopping

Early stopping is a form of regularization while training a model with an iterative method, such as gradient descent. Since all the neural networks learn exclusively by using gradient descent, early stopping is a technique applicable to all the problems. This method update the model so as to make it better fit the training data with each iteration. Up to a point, this improves the model's performance on data on the test set. Past that point however, improving the model's fit to the training data leads to increased generalization error. Early stopping rules provide guidance as to how many iterations can be run before the model begins to overfit.

min_delta: The minimum change in the monitored metric to qualify as an improvement, i.e., an absolute change less than min_delta will count as no improvement.

patience: Number of epochs with no improvement after which training will be stopped.

mode: Defines what should be considered as an improvement, depending on the monitored metric. Commonly set to 'min' for loss where a decrease is better.

```
model.fit(x_train, y_train, epochs=100, validation_data=(x_val, y_val), callbacks=[early_stopping])
```

3. Use Data Augmentation

In the case of neural networks, data augmentation simply means increasing size of the data that is increasing the number of images present in the dataset. Some of the popular image augmentation techniques are flipping, translation, rotation, scaling, changing brightness, adding noise etcetera This technique is shown in the diagram. As we can see, using data augmentation a lot of similar images can be generated. This helps in increasing the dataset size and thus reduce overfitting. The reason is that, as we add more data, the model is unable to overfit all the samples, and is forced to generalize.

4. Use Regularization

Regularization is a technique to reduce the complexity of the model. It does so by adding a penalty term to the loss function. The most common techniques are known as L1 and L2 regularization:

The L1 penalty aims to minimize the absolute value of the weights. This is mathematically shown in the below formula.

$$\boxed{L(x,y) \equiv \sum_{i=1}^{n}(y_i - h_\theta(x_i))^2 + \boxed{\lambda \sum_{i=1}^{n}|\theta_i|}}$$

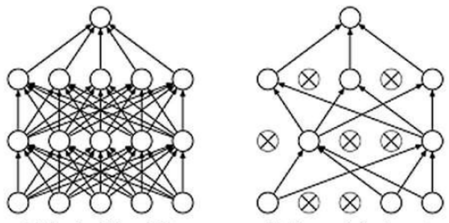
The L2 penalty aims to minimize the squared magnitude of the weights. This is mathematically shown in the below formula.

$$\boxed{L(x,y) \equiv \sum_{i=1}^{n}(y_i - h_\theta(x_i))^2 + \boxed{\lambda \sum_{i=1}^{n}\theta_i^2}}$$

```
layer = layers.Dense(units=5,

                     kernel_initializer='ones',

                     kernel_regularizer=regularizers.L1(0.01),

                     activity_regularizer=regularizers.L2(0.01))
```

5. Use Dropouts

Dropout is a regularization technique that prevents neural networks from overfitting. Regularization methods like L1 and L2 reduce overfitting by modifying the cost function. Dropout on the other hand, modify the network itself. It randomly drops neurons from the neural network during training in each iteration. When we drop different sets of neurons, it's equivalent to training different neural networks. The different networks will overfit in different ways, so the net effect of dropout will be to reduce overfitting.


(a) Standard Neural Net (b) After applying dropout.