Python Tutorial and Homework 2

This Colab Notebook contains 2 sections. The first section is a Python Tutorial intended to make you familiar with Numpy and Colab functionalities. This section will not be graded.

The second section is a coding assignment (Homework 2). This section will be graded

1.0 Python Tutorial

This assignment section won't be graded but is intended as a tutorial to refresh the basics of python and its dependencies. It also allows one to get familiarized with Google Colab.

1.0.0 Array manipulation using numpy

Q1 - Matrix multiplication

In [5]:

```
import numpy as np

### Create two numpy arrays with the dimensions 3x2 and 2x3 respectively using  np.arange
().
### The elements of the vector are
### Vector 1 elements = [ 2,   4,   6,   8, 10, 12];
### Vector 2 elements = [ 7, 10, 13, 16, 19, 22]

### Starting at 2, stepping by 2
vector1 = np.array([[2,  4], [ 6,  8], [10, 12]])
### Starting at 7, stepping by 3
vector2 =np.array([[7, 10, 13], [16, 19, 22]])

### Print vec
# print(vector1, vector2)
print(vector1)
print(vector2)
### Take product of the two matricies (Matrix product)
prod = np.dot(vector1,vector2)

### Print
print(prod)
```

```
[[ 2  4]
 [ 6  8]
 [10 12]]
[[ 7 10 13]
 [16 19 22]]
[[ 78  96 114]
 [170 212 254]
 [262 328 394]]
```

Q2 - Diagonals

In [16]:

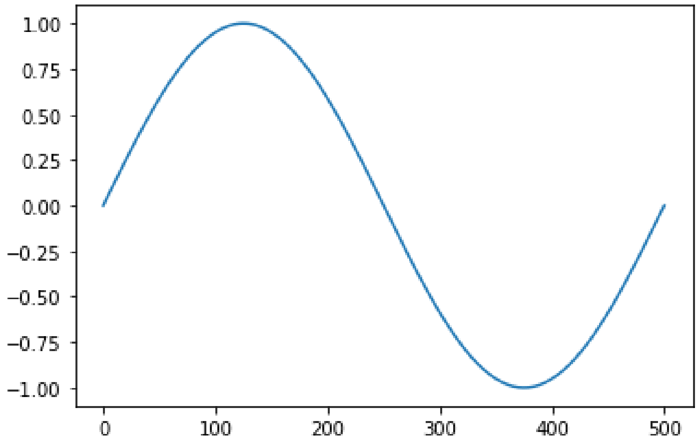
```
### Create two numpy arrays with the dimensions 10x10 using the function np.arange().
### Starting at 2, stepping by 3
vector1 = np.arange(2, 302, 3).reshape(10, 10)
```

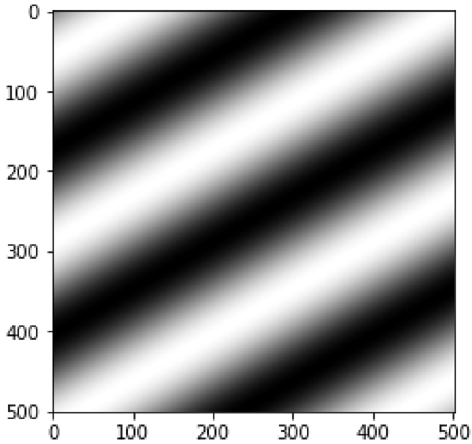
```python
### Starting at 35, stepping by 9
vector2 = np.arange(35, 935, 9).reshape(10, 10)

### Print vec
print(vector1)
print(vector2)
### Obtain the diagonal matrix of each vector1 such that the start of the diagonal is fro
m (3,0) and the end is (9,6)
### Reshape the the matrix such that it form a diagonal maritix of shape(7,7)

vector1_offset_diagonal  = np.diag(np.diag(vector1[3:10, 0:7]))
print(vector1_offset_diagonal)
### Obtain a 7x7 matrix from the vector 2
### starting from (left top element) = (0,3)
### ending at (right bottom element) = (6,9)
vector2_offset_diagonal = np.diag(np.diag(vector2[0:7, 3:10]))
print(vector2_offset_diagonal)
### Print diagonal matrix
# print(vector1_offset_diagonal, vector2_offset_diagonal)


### Take product of the two diagonal matricies (Matrix product)
prod = np.dot(vector1_offset_diagonal,vector2_offset_diagonal)

### Print
print(prod)
```

```
[[  2   5   8  11  14  17  20  23  26  29]
 [ 32  35  38  41  44  47  50  53  56  59]
 [ 62  65  68  71  74  77  80  83  86  89]
 [ 92  95  98 101 104 107 110 113 116 119]
 [122 125 128 131 134 137 140 143 146 149]
 [152 155 158 161 164 167 170 173 176 179]
 [182 185 188 191 194 197 200 203 206 209]
 [212 215 218 221 224 227 230 233 236 239]
 [242 245 248 251 254 257 260 263 266 269]
 [272 275 278 281 284 287 290 293 296 299]]
[[ 35  44  53  62  71  80  89  98 107 116]
 [125 134 143 152 161 170 179 188 197 206]
 [215 224 233 242 251 260 269 278 287 296]
 [305 314 323 332 341 350 359 368 377 386]
 [395 404 413 422 431 440 449 458 467 476]
 [485 494 503 512 521 530 539 548 557 566]
 [575 584 593 602 611 620 629 638 647 656]
 [665 674 683 692 701 710 719 728 737 746]
 [755 764 773 782 791 800 809 818 827 836]
 [845 854 863 872 881 890 899 908 917 926]]
[[ 92   0   0   0   0   0   0]
 [  0 125   0   0   0   0   0]
 [  0   0 158   0   0   0   0]
 [  0   0   0 191   0   0   0]
 [  0   0   0   0 224   0   0]
 [  0   0   0   0   0 257   0]
 [  0   0   0   0   0   0 290]]
[[ 62   0   0   0   0   0   0]
 [  0 161   0   0   0   0   0]
 [  0   0 260   0   0   0   0]
 [  0   0   0 359   0   0   0]
 [  0   0   0   0 458   0   0]
 [  0   0   0   0   0 557   0]
 [  0   0   0   0   0   0 656]]
[[  5704      0      0      0      0      0      0]
 [     0  20125      0      0      0      0      0]
 [     0      0  41080      0      0      0      0]
 [     0      0      0  68569      0      0      0]
 [     0      0      0      0 102592      0      0]
 [     0      0      0      0      0 143149      0]
 [     0      0      0      0      0      0 190240]]
```

Q3 - Sin wave

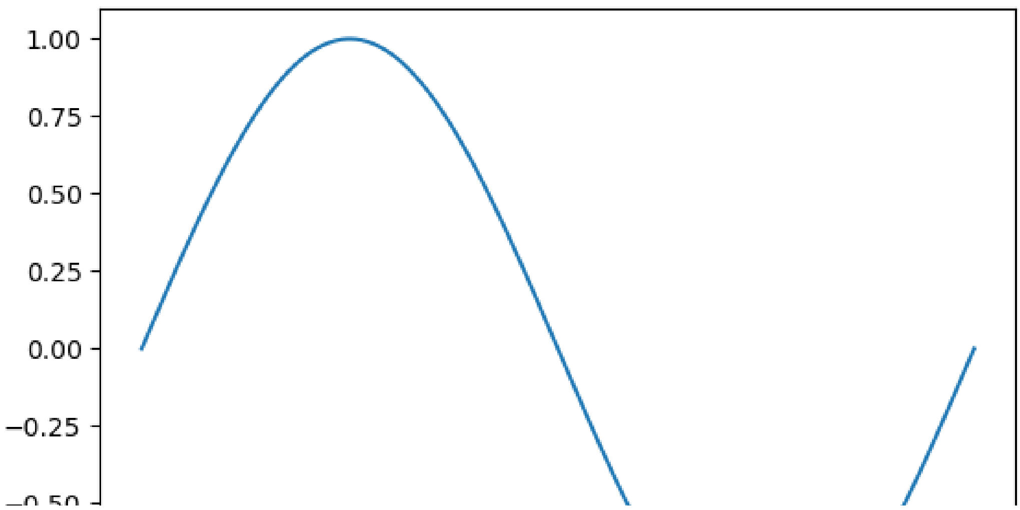
Sample outputs,

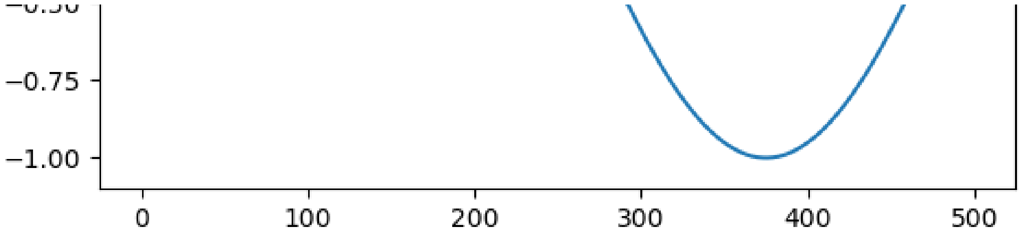




In [18]:

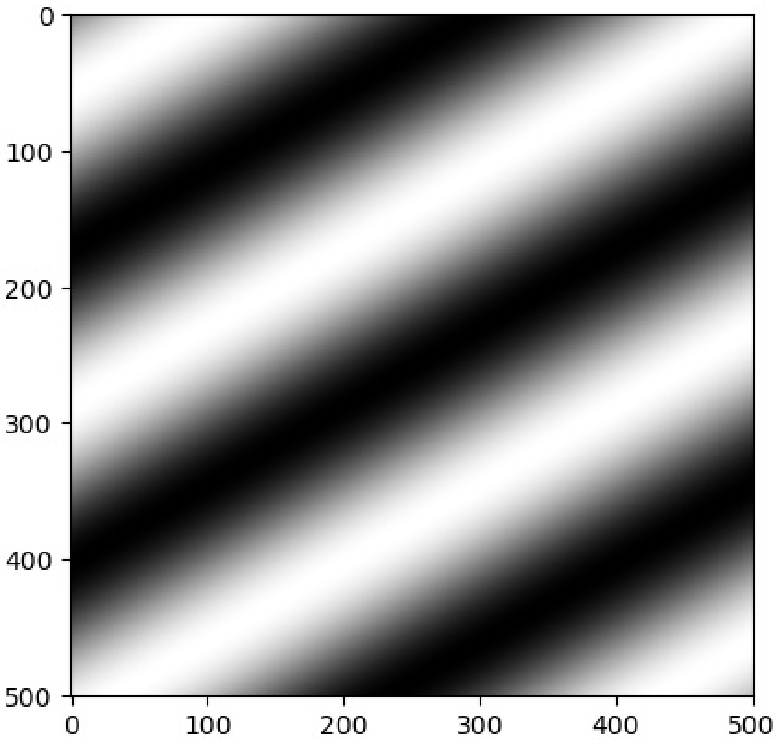
```
import matplotlib.pyplot as plt
import numpy as np
### Create a time matrix that evenly samples a sine wave at a frequency of 1Hz
### Starting at time step T = 0
### End at time step T = 500

freq = 1   # 1Hz
time_steps = 500
time = np.arange(0, time_steps+1, 1)

### Given wavelength of
wavelength = 500

### Construct a sin wave using the formula sin(2*pi*(time/wavelength))
y =np.sin(2 * np.pi * (time / wavelength))

#### Plot the wave
plt.plot(time, y)
plt.show()
```





In [22]:

```
#### Given a 2D mesh grid
X, Y = np.meshgrid(time, time)
#### wavelength and angle of rotation(phi) of the sin wave in 2d. Imagine a 2D sine wave
is being rotating about the Z axes
wavelength = 200
phi = np.pi / 3

#### Calculate the sin wave in 2d space using the formula sin(2*pi*(x'/wavelength)) where
x'=Xcos(phi) + Ysin(phi)
x_bar = X * np.cos(phi) + Y * np.sin(phi)
grating = np.sin(2*np.pi*(x_bar/wavelength))

#### Plot the wave
#### Intuition, think of the white area as hills and the black areas as valleys
plt.set_cmap("gray")
plt.imshow(grating)
plt.show()
```



Q4 Car Brands

In [6]:

```
cars = ['Civic', 'Insight', 'Fit', 'Accord', 'Ridgeline', 'Avancier','Pilot', 'Legend',
'Beat', 'FR-V', 'HR-V', 'Shuttle']

#### Create a 3D array of cars of shape 2,3,2
cars_3d = np.array(cars).reshape(2,3,2)
print(cars_3d)
#### Extract the top layer of the matrix. Top layer of a matrix A of shape(2,3,2) will ha
ve the following structue A_top = [[A[0,0,0], A[0,0,1]],[A[0,1,0],A[0,1,1]],[A[0,2,0],A[0
,2,1]]]
#### HINT - Array slicing or splitting
```

```python
cars_top_layer = cars_3d[0]

#### Similarly extract the bottom layer
#### HINT - Array slicing or splitting
cars_bottom_layer = cars_3d[1]

#### Print layers
print("\nTop Layer \n ",cars_top_layer,"\nBottom Layer\n", cars_bottom_layer)

#### Flatten the top layer
cars_top_flat = cars_top_layer.flatten()
#### Flatten the bottom layer
cars_bottom_flat = cars_bottom_layer.flatten()

#### Print layers
print("\nTop Flattened : ",cars_top_flat,"\nBottom Flattened : ",cars_bottom_flat)


new_car_list = np.empty((cars_top_layer.size + cars_bottom_layer.size,), dtype=object)


#### Interweave the to flattened lists and insert into new_car_list such that new_car_lis
t=['Civic' 'Pilot' 'Fit' 'Beat' 'Ridgeline' 'HR-V' 'Insight' 'Legend' 'Accord' 'FR-V' 'Av
ancier' 'Shuttle']
#### Using only array slicing
new_car_list[0:2] = [cars_top_flat[0],cars_bottom_flat[0]]
new_car_list[2:4] = [cars_top_flat[2],cars_bottom_flat[2]]
new_car_list[4:6] = [cars_top_flat[4],cars_bottom_flat[4]]
new_car_list[6:8] = [cars_top_flat[1],cars_bottom_flat[1]]
new_car_list[8:10] =[cars_top_flat[3],cars_bottom_flat[3]]
new_car_list[10:12] = [cars_top_flat[5],cars_bottom_flat[5]]
print(new_car_list)

#### Concatenate and flatten the top and bottom layer such that the final list is of the
form cat_flat = ['Civic' 'Insight' 'Pilot' 'Legend' 'Fit' 'Accord' 'Beat' 'FR-V' 'Ridgeli
ne' 'Avancier' 'HR-V' 'Shuttle']
car_flat = np.empty((cars_top_layer.size + cars_bottom_layer.size,), dtype=object)
car_flat[0:2] = [cars_top_flat[0],cars_top_flat[1]]
car_flat[2:4] = [cars_bottom_flat[0],cars_bottom_flat[1]]
car_flat[4:6] = [cars_top_flat[2],cars_top_flat[3]]
car_flat[6:8] = [cars_bottom_flat[2],cars_bottom_flat[3]]
car_flat[8:10] = [cars_top_flat[4],cars_top_flat[5]]
car_flat[10:12] = [cars_bottom_flat[4],cars_bottom_flat[5]]


#### Print layers
print("\n\nInterwoven - ", new_car_list,"\nConcatenate and flatten - ", car_flat)
```

```
[[['Civic' 'Insight']
  ['Fit' 'Accord']
  ['Ridgeline' 'Avancier']]

 [['Pilot' 'Legend']
  ['Beat' 'FR-V']
  ['HR-V' 'Shuttle']]]

Top Layer
  [['Civic' 'Insight']
 ['Fit' 'Accord']
 ['Ridgeline' 'Avancier']]
Bottom Layer
 [['Pilot' 'Legend']
 ['Beat' 'FR-V']
 ['HR-V' 'Shuttle']]

Top Flattened :  ['Civic' 'Insight' 'Fit' 'Accord' 'Ridgeline' 'Avancier']
Bottom Flattened :  ['Pilot' 'Legend' 'Beat' 'FR-V' 'HR-V' 'Shuttle']
['Civic' 'Pilot' 'Fit' 'Beat' 'Ridgeline' 'HR-V' 'Insight' 'Legend'
 'Accord' 'FR-V' 'Avancier' 'Shuttle']


Interwoven -  ['Civic' 'Pilot' 'Fit' 'Beat' 'Ridgeline' 'HR-V' 'Insight' 'Legend'
```

```
 'Accord' 'FR-V' 'Avancier' 'Shuttle']
Concatenate and flatten -  ['Civic' 'Insight' 'Pilot' 'Legend' 'Fit' 'Accord' 'Beat' 'FR-V'
 'Ridgeline' 'Avancier' 'HR-V' 'Shuttle']
```

1.0.1 Basics tensorflow

Helper functions

In [7]:

```
import tensorflow as tf
from tensorflow import keras
import numpy as np
import matplotlib.pyplot as plt
```

```
F:\Anaconda\envs\testenv\lib\site-packages\tensorflow\python\framework\dtypes.py:516: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint8 = np.dtype([("qint8", np.int8, 1)])
F:\Anaconda\envs\testenv\lib\site-packages\tensorflow\python\framework\dtypes.py:517: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_quint8 = np.dtype([("quint8", np.uint8, 1)])
F:\Anaconda\envs\testenv\lib\site-packages\tensorflow\python\framework\dtypes.py:518: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint16 = np.dtype([("qint16", np.int16, 1)])
F:\Anaconda\envs\testenv\lib\site-packages\tensorflow\python\framework\dtypes.py:519: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_quint16 = np.dtype([("quint16", np.uint16, 1)])
F:\Anaconda\envs\testenv\lib\site-packages\tensorflow\python\framework\dtypes.py:520: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint32 = np.dtype([("qint32", np.int32, 1)])
F:\Anaconda\envs\testenv\lib\site-packages\tensorflow\python\framework\dtypes.py:525: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  np_resource = np.dtype([("resource", np.ubyte, 1)])
F:\Anaconda\envs\testenv\lib\site-packages\tensorboard\compat\tensorflow_stub\dtypes.py:541: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint8 = np.dtype([("qint8", np.int8, 1)])
F:\Anaconda\envs\testenv\lib\site-packages\tensorboard\compat\tensorflow_stub\dtypes.py:542: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_quint8 = np.dtype([("quint8", np.uint8, 1)])
F:\Anaconda\envs\testenv\lib\site-packages\tensorboard\compat\tensorflow_stub\dtypes.py:543: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint16 = np.dtype([("qint16", np.int16, 1)])
F:\Anaconda\envs\testenv\lib\site-packages\tensorboard\compat\tensorflow_stub\dtypes.py:544: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_quint16 = np.dtype([("quint16", np.uint16, 1)])
F:\Anaconda\envs\testenv\lib\site-packages\tensorboard\compat\tensorflow_stub\dtypes.py:545: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint32 = np.dtype([("qint32", np.int32, 1)])
F:\Anaconda\envs\testenv\lib\site-packages\tensorboard\compat\tensorflow_stub\dtypes.py:550: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  np_resource = np.dtype([("resource", np.ubyte, 1)])
```

In [8]:

```
def plot_image(i, predictions_array, true_label, img):
  true_label, img = true_label[i], img[i]
```

```
  plt.grid(False)
  plt.xticks([])
  plt.yticks([])

  plt.imshow(img, cmap=plt.cm.binary)

  predicted_label = np.argmax(predictions_array)
  if predicted_label == true_label:
    color = 'blue'
  else:
    color = 'red'

def plot_value_array(i, predictions_array, true_label):
  true_label = true_label[i]
  plt.grid(False)
  plt.xticks(range(10))
  plt.yticks([])
  thisplot = plt.bar(range(10), predictions_array, color="#777777")
  plt.ylim([0, 1])
  predicted_label = np.argmax(predictions_array)

  thisplot[predicted_label].set_color('red')
  thisplot[true_label].set_color('blue')
```

Q1 MNIST Classifier

In [9]:

```
## Import the MNIST dataset from keras
mnist = tf.keras.datasets.mnist
### Load the data
(x_train, y_train), (x_test, y_test) = mnist.load_data()

### Normalize the 8bit images with values in the range [0,255]
x_train, x_test = x_train/255.0, x_test/255.0
```

In [11]:

```
from tensorflow.keras import layers, models
## Create a model with the following architecture Flatten -> Dense(128, relu) -> Dense(64
,relu) -> outpuLayer(size=10)
model = models.Sequential()

# Flatten layer
model.add(layers.Flatten(input_shape=(28, 28)))  #image size 28x28

# Dense layer with 128 units and ReLU activation
model.add(layers.Dense(128, activation='relu'))

# Dense layer with 64 units and ReLU activation
model.add(layers.Dense(64, activation='relu'))

# Output layer with 10 units (assuming 10 classes)
model.add(layers.Dense(10, activation='relu'))

### Complie the model with the adam optimizer and crossentropy loss
### HINT - No One hot encoding

model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accurac
y'])

# Display the model summary
model.summary()
```

```
WARNING:tensorflow:From F:\Anaconda\envs\testenv\lib\site-packages\tensorflow\python\ops\
init_ops.py:1251: calling VarianceScaling.__init__ (from tensorflow.python.ops.init_ops)
with dtype is deprecated and will be removed in a future version.
Instructions for updating:
Call initializer instance with the dtype argument instead of passing it to the constructo
r
Model: "sequential"
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
flatten (Flatten)            (None, 784)               0         
_________________________________________________________________
dense (Dense)                (None, 128)               100480    
_________________________________________________________________
dense_1 (Dense)              (None, 64)                8256      
_________________________________________________________________
dense_2 (Dense)              (None, 10)                650       
=================================================================
Total params: 109,386
Trainable params: 109,386
Non-trainable params: 0
_________________________________________________________________
```

In [12]:

```
### Train the model on the train data for 5 epochs
model.fit(x_train, y_train, epochs=5)
```

```
WARNING:tensorflow:From F:\Anaconda\envs\testenv\lib\site-packages\tensorflow\python\ops\
math_grad.py:1250: add_dispatch_support.<locals>.wrapper (from tensorflow.python.ops.arra
y_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
Epoch 1/5
60000/60000 [==============================] - 8s 142us/sample - loss: 2.3524 - acc: 0.10
70
Epoch 2/5
60000/60000 [==============================] - 8s 135us/sample - loss: 2.3024 - acc: 0.09
91
Epoch 3/5
60000/60000 [==============================] - 7s 121us/sample - loss: 2.3026 - acc: 0.09
87
Epoch 4/5
60000/60000 [==============================] - 8s 132us/sample - loss: 2.3026 - acc: 0.09
87
Epoch 5/5
60000/60000 [==============================] - 7s 121us/sample - loss: 2.3026 - acc: 0.09
87
```

Out[12]:

```
<tensorflow.python.keras.callbacks.History at 0x1d287684988>
```

In [34]:

```
### Check the accuracy of the trained model
test_loss, test_acc = model.evaluate(x_test,  y_test, verbose=2)
print('\nTest accuracy:', test_acc)
```

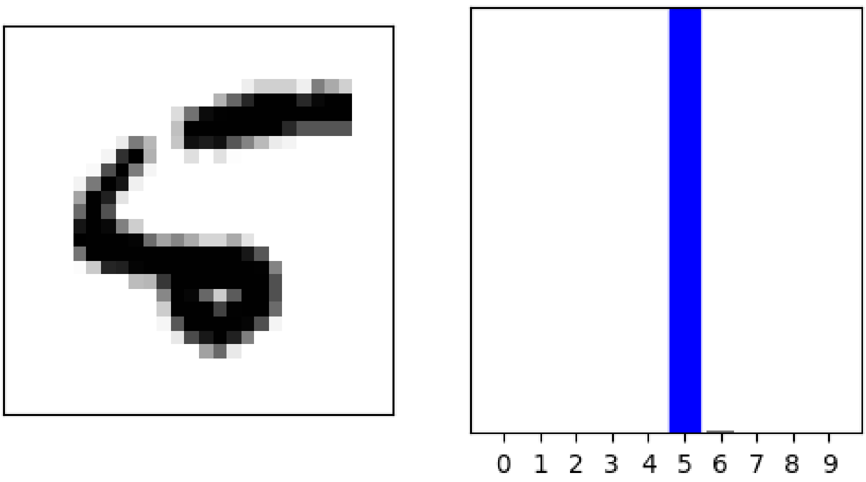
```
10000/10000 - 0s - loss: 2.3026 - acc: 0.0980

Test accuracy: 0.098
```

In [13]:

```
### Convert the above model to a probabilistic model with a softmax as the output layer
probability_model = models.Sequential()

# Flatten layer
probability_model.add(layers.Flatten(input_shape=(28, 28)))  #image size 32x64

# Dense layer with 128 units and ReLU activation
probability_model.add(layers.Dense(128, activation='relu'))

# Dense layer with 64 units and ReLU activation
probability_model.add(layers.Dense(64, activation='relu'))

# Output layer with 10 units (assuming 10 classes)
probability_model.add(layers.Dense(10, activation='softmax'))
```

```python
probability_model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

### Train the model on the train data for 5 epochs
probability_model.fit(x_train, y_train, epochs=5,)

### Check the accuracy of the trained model
test_loss, test_acc = probability_model.evaluate(x_test,  y_test, verbose=2)
print('\nTest accuracy:', test_acc)

### Run the test data through the new model and get predictions
predictions = probability_model.predict(x_test)

### Plot a test output
i = 8### <---- Change this to some random number to see different predictions
plt.figure(figsize=(6,3))
plt.subplot(1,2,1)
plot_image(i, predictions[i], y_test, x_test)
plt.subplot(1,2,2)
plot_value_array(i, predictions[i],  y_test)
plt.show()
### Blue bars mean correct guess red bar means wrong guess!!
```

```
Epoch 1/5
60000/60000 [==============================] - 8s 132us/sample - loss: 0.2445 - acc: 0.9284
Epoch 2/5
60000/60000 [==============================] - 8s 135us/sample - loss: 0.1064 - acc: 0.9682
Epoch 3/5
60000/60000 [==============================] - 7s 125us/sample - loss: 0.0765 - acc: 0.9768
Epoch 4/5
60000/60000 [==============================] - 8s 130us/sample - loss: 0.0564 - acc: 0.9828
Epoch 5/5
60000/60000 [==============================] - 7s 121us/sample - loss: 0.0450 - acc: 0.9856
10000/10000 - 1s - loss: 0.0797 - acc: 0.9777

Test accuracy: 0.9777
```



1.0.2 Basic Pytorch Tutorial

In [14]:

```python
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms
```

```python
# Set the random seed for reproducibility
torch.manual_seed(42)

# Define a simple feedforward neural network
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(784, 128)  # 28x28 input size (MNIST images are 28x28 pixels)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(128, 10)   # 10 output classes (digits 0-9)

    def forward(self, x):
        x = x.view(-1, 784)  # Flatten the input image
        x = self.fc1(x)
        x = self.relu(x)
        x = self.fc2(x)
        return x

# Load the MNIST dataset and apply transformations
transform = transforms.Compose([transforms.ToTensor(), transforms.Normalize((0.5,), (0.5,))])

trainset = torchvision.datasets.MNIST(root='./data', train=True, download=True, transform=transform)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=64, shuffle=True)

testset = torchvision.datasets.MNIST(root='./data', train=False, download=True, transform=transform)
testloader = torch.utils.data.DataLoader(testset, batch_size=64, shuffle=False)

# Initialize the neural network and optimizer
net = Net()
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.01)

# Training loop
num_epochs = 10
for epoch in range(num_epochs):
    running_loss = 0.0
    for i, data in enumerate(trainloader, 0):
        inputs, labels = data

        optimizer.zero_grad()

        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        running_loss += loss.item()

    print(f'Epoch {epoch+1}, Loss: {running_loss / len(trainloader)}')

print('Finished Training')

# Evaluate the model on the test set
correct = 0
total = 0
with torch.no_grad():
    for data in testloader:
        images, labels = data
        outputs = net(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print(f'Accuracy on test set: {100 * correct / total}%')
```

```
Epoch 1, Loss: 0.7497774055485786
Epoch 2, Loss: 0.3669378405758567
Epoch 3, Loss: 0.32100593225597573
```

```
Epoch 4, Loss: 0.2938191123656245
Epoch 5, Loss: 0.27316678917881393
Epoch 6, Loss: 0.25377058785067186
Epoch 7, Loss: 0.23668134354674486
Epoch 8, Loss: 0.2213350784367145
Epoch 9, Loss: 0.2073744827114951
Epoch 10, Loss: 0.19491218166278879
Finished Training
Accuracy on test set: 94.45%
```

2.0 Homework 2

90 points

Note : This section will be graded and must be attemped using Pytorch only

Graded Section : Deep Learning Approach

Time-Series Prediction Time series and sequence prediction could be a really amazing to predict/estimate a robot's trajectory which requires temporal data at hand. In this assignemnt we will see how this could be done using Deep Learning.

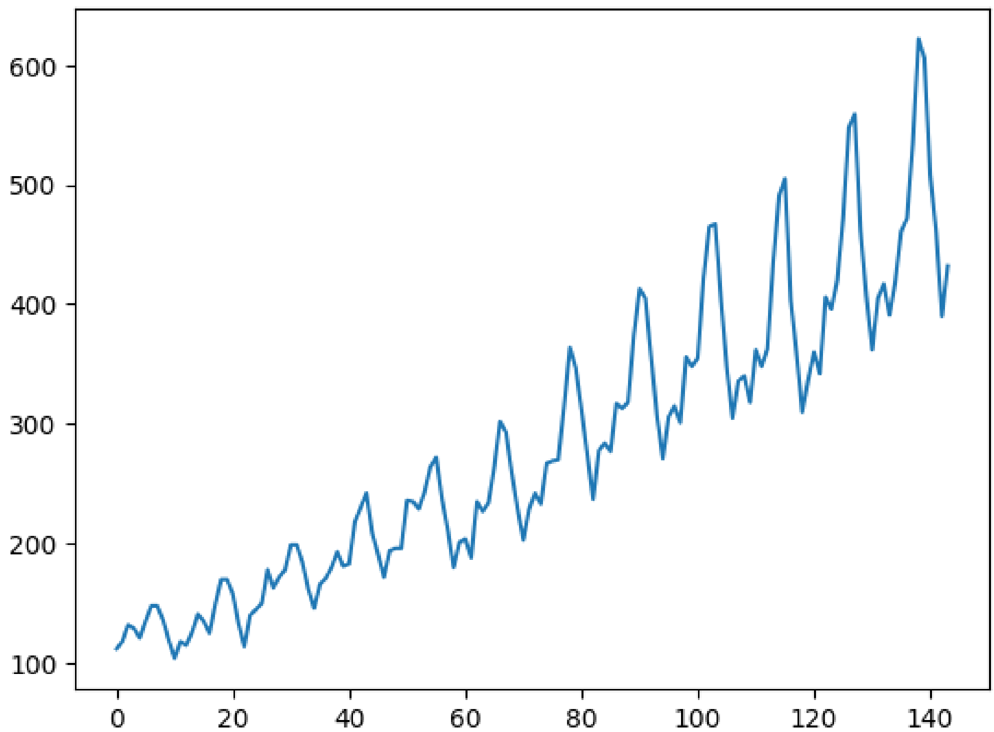
Given a dataset link for airline passengers prediction problem. Predict the number of international airline passengers in units of 1,000 given a year and a month. Here is how the data looks like.

In [190]:

```
import matplotlib.pyplot as plt
import pandas as pd
file_name = 'F:/Robot_learning/airline.csv' # dataset path
# Reading data using pandas or csv
df = pd.read_csv(file_name)
```

In [191]:

```
# plotting the dataset
timeseries = df[["Passengers"]].values.astype('float32')
# plotting the dataset
plt.plot(timeseries)
plt.show()
```



1. **Write the dataloader code to pre-process the data for pytorch tensors using any library of your choice. Here is a good resource for the dataloader [Video link]()**

In [192]:

```
#Write your code here for the dataloader in Pytorch

#Importing all necessary libs
import torch
import torch.nn as nn
import torch.optim as optim
import torch.utils.data as data
import numpy as np
import matplotlib.pyplot as plt

timeseries = df[["Passengers"]].values.astype('float32')
train_size = int(len(df) * 0.7)
test_size = int(len(df) - train_size)
train = timeseries[:train_size]
test = timeseries[train_size:]

def create_sequences(data, seq_length):
    xs, ys = [], []
    for i in range(len(data) - seq_length):
        x = data[i:(i + seq_length)]
        y = data[i+1:i + seq_length+1]
        xs.append(x)
        ys.append(y)
    return torch.tensor(xs), torch.tensor(ys)

# Preprocess the historical data
#Defining look back window size
window_size = 8
X_train, y_train = create_sequences(train, window_size)
X_test , y_test  = create_sequences(test, window_size)

#Dataloader
batch_size = 5
dataloader = data.DataLoader(data.TensorDataset(X_train, y_train), shuffle=True, batch_s
ize=batch_size)
```

2. Create the model in pytorch here uinsg 1. Long-Short Term Memory (LSTM) and 2. Recurrent Neural Network (RNN). Here is a good resource for Custom model generation.

Train using the two models. Here is the resource for the same [Video link]()

In [193]:

```
#LSTM Model class
class lstm(nn.Module):
  #constructor
  def __init__(self):
    super().__init__()
    self.lstm = nn.LSTM(input_size=1, hidden_size=50, num_layers=1, batch_first=True)
    self.linear = nn.Linear(50, 1)
  def forward(self, val):
    val, _ = self.lstm(val)
    val    = self.linear(val)
    return val

#Setting parameters
input_size = 1
num_layers = 1
hidden_size = 50
output_size = 1

#LSTM Model
```

```python
model = lstm()

#Adam optimizer and Loss
optimizer = optim.Adam(model.parameters())
criterion = nn.MSELoss()

#Training for Batchsize and epochs
total_epochs = 1000
for epoch in range(total_epochs):
  model.train()
  for i, j in dataloader:
    optimizer.zero_grad()
    y_pred = model(i)
    loss = criterion(y_pred, j)
    loss.backward()
    optimizer.step()
  model.eval()
  with torch.no_grad():
    if (epoch % 100 == 0):
        y_pred = model(X_test)
        testMSE_LSTM = np.sqrt(criterion(y_pred, y_test))
        print(f"Epoch [{epoch}/{total_epochs}], Loss: {loss.item():.4f}")

with torch.no_grad():
    trainLSTM = np.ones_like(timeseries) * np.nan
    y_predLSTM = model(X_train)
    y_predLSTM = y_predLSTM[:, -1, :]
    trainLSTM[window_size:train_size] = model(X_train)[:, -1, :]
    testLSTM = np.ones_like(timeseries) * np.nan
    testLSTM[train_size+window_size:len(timeseries)] = model(X_test)[:, -1, :]
```

```
Epoch [0/1000], Loss: 47146.1953
Epoch [100/1000], Loss: 31385.6523
Epoch [200/1000], Loss: 4887.8999
Epoch [300/1000], Loss: 989.6507
Epoch [400/1000], Loss: 5364.0508
Epoch [500/1000], Loss: 309.8328
Epoch [600/1000], Loss: 455.2224
Epoch [700/1000], Loss: 566.1150
Epoch [800/1000], Loss: 388.8507
Epoch [900/1000], Loss: 110.0214
```

In [194]:

```python
#Class for RNN model
class rnn(nn.Module):
  def __init__(self):
    super().__init__()
    self.rnn = nn.RNN(input_size=1, hidden_size=50, num_layers=1, batch_first=True)
    self.linear = nn.Linear(50, 1)
  def forward(self, val):
    val, _ = self.rnn(val)
    val = self.linear(val)
    return val

#Setting parameters
input_size = 1
num_layers = 1
hidden_size = 50
output_size = 1

#RNN Model
model = rnn()

#Adam optimizer and Loss
optimizer = optim.Adam(model.parameters())
criterion = nn.MSELoss()

#Training for Batchsize and epochs
total_epochs = 1000
for epoch in range(total_epochs):
  model.train()
```

```
  for i, j in dataloader:
    optimizer.zero_grad()
    y_pred = model(i)
    loss = criterion(y_pred, j)
    loss.backward()
    optimizer.step()
  model.eval()

  with torch.no_grad():
    if (epoch % 100 == 0):
        y_pred = model(X_test)
        testMSE_RNN = np.sqrt(criterion(y_pred, y_test))
        print(f"Epoch [{epoch}/{total_epochs}], Loss: {loss.item():.4f}")

with torch.no_grad():
    trainRNN = np.ones_like(timeseries) * np.nan
    y_predRNN = model(X_train)
    y_predRMM = y_predRNN[:, -1, :]
    trainRNN[window_size:train_size] = model(X_train)[:, -1, :]
    testRNN = np.ones_like(timeseries) * np.nan
    testRNN[train_size+window_size:len(timeseries)] = model(X_test)[:, -1, :]
```

```
Epoch [0/1000], Loss: 28802.4180
Epoch [100/1000], Loss: 8374.3154
Epoch [200/1000], Loss: 5047.5801
Epoch [300/1000], Loss: 6307.8149
Epoch [400/1000], Loss: 258.3547
Epoch [500/1000], Loss: 343.7880
Epoch [600/1000], Loss: 172.3636
Epoch [700/1000], Loss: 159.8618
Epoch [800/1000], Loss: 248.8396
Epoch [900/1000], Loss: 427.5971
```

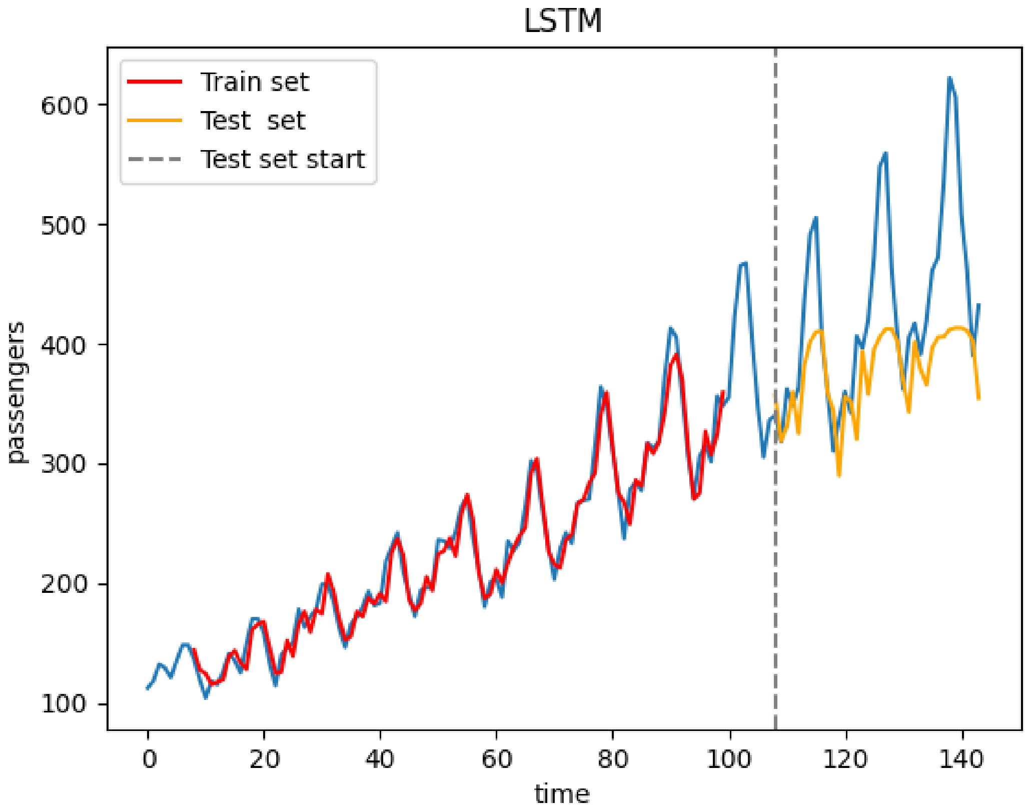
1. **Evaluate and Compare the result using proper metric. Justify the metrics used.**

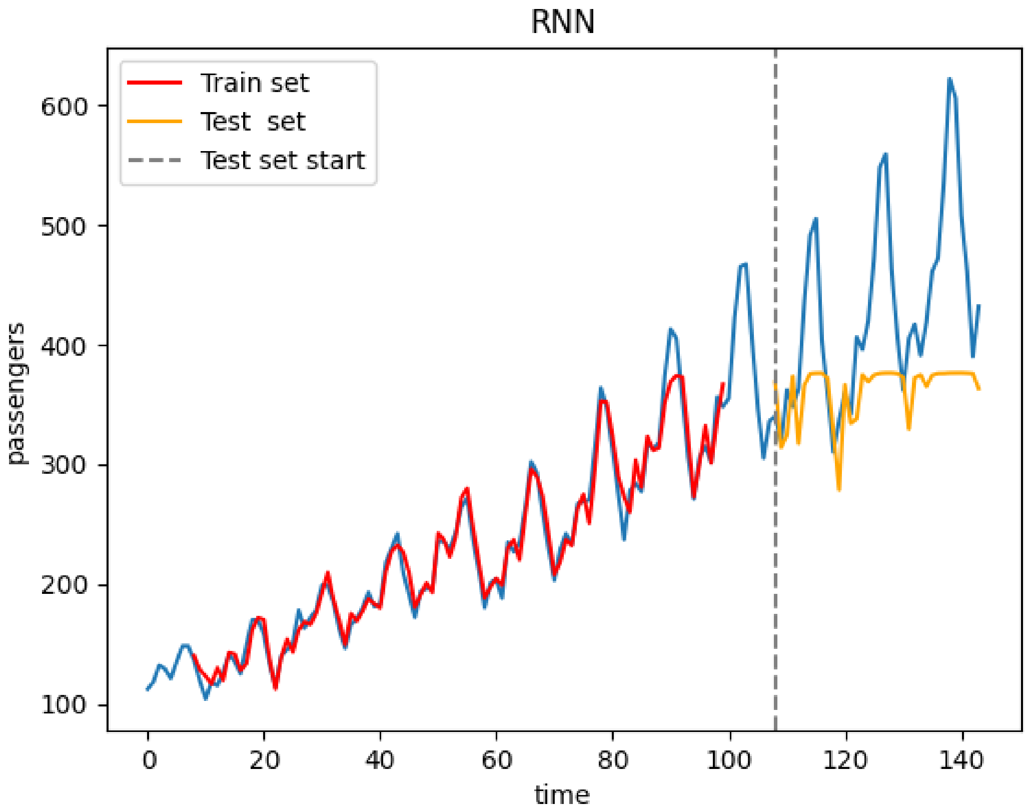
In [195]:

```
print("\nEpoch:",1000, "lstm-RMSE--Test",testMSE_LSTM)

print("\nEpoch:",1000, "RNN-RMSE--Test",testMSE_RNN)


#LSTM
# Calculate the index where the test set starts
test_start_index = len(timeseries) - len(y_test) - window_size

plt.plot(timeseries)
plt.plot(trainLSTM, color='red',   label='Train set')
plt.plot(testLSTM, color='orange', label='Test  set')
plt.axvline(x=test_start_index+window_size, color='gray', linestyle='--', label="Test se
t start")
plt.title('LSTM')
plt.xlabel('time')
plt.ylabel('passengers')
plt.legend()
plt.show()


#RNN
# Calculate the index where the test set starts
test_start_index = len(timeseries) - len(y_test) - window_size

plt.plot(timeseries)
plt.plot(trainRNN, color='red',   label='Train set')
plt.plot(testRNN,  color='orange', label='Test  set')
plt.axvline(x=test_start_index+window_size, color='gray', linestyle='--', label="Test se
t start")
plt.title('RNN')
plt.xlabel('time')
plt.ylabel('passengers')
```

```
plt.legend()
plt.show()
```

```
Epoch: 1000 lstm-RMSE--Test tensor(87.4660)

Epoch: 1000 RNN-RMSE--Test tensor(92.7797)
```





1.The root mean squared value (RMSE) for LSTM is lower than that of RNN, it indicates that the LSTM model is providing more accurate predictions on the given dataset. Lower RMSE signifies that the model's predictions are closer to the actual values, suggesting better performance in terms of prediction accuracy. 2.Therefore for the dataset provided LSTM has better predictions.

[Bonus 5 points] Suggest some things that could be done to improve the results.

1.Running the model on GPU instead of CPU 2.Increase Model Complexity 3.Apply gradient clipping to prevent

exploding gradients during training. 4.Reduce the batch size to introduce more noise. 5.Using nn.Dropout layer after each LSTM layer to prevent overfitting.

[Bonus 5 points] Suggest where this could be used in Robotics other than the example given in the beginning.

1.LSTMs can be used to predict the trajectories of moving objects in a robot's environment. 2.RNNs and LSTMs can aid in SLAM tasks by predicting the next state of the robot based on past sensor measurements, improving real-time mapping and localization accuracy. 3.Object detection and Grasping based on past events. 4.LSTM and RNN can be used alongside reinforcement learning.