RNN using MNIST

- MNIST data set can be treated as sequence of rows and columns of pixels.
- Process MNIST image as 28-element input vector and timesteps equal to 28.

```
# import the library
import numpy as np
from keras.models import Sequential
from keras.layers import Dense, Activation , SimpleRNN
from keras.utils import to_categorical
from keras.datasets import mnist
```

```
# load mnist dataset
(x_train,y_train), (x_test,y_test) = mnist.load_data()
print(x_train.shape)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 0s 0us/step
(60000, 28, 28)
```

```
# compute the number of labels
num_labels = len(np.unique(y_train)) # len returns length
num_labels
```

```
10
```

```
# convert to categorical
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
```

```
# normalise
x_train = x_train.astype('float')/255
x_test = x_test.astype('float')/255
```

```
# network parameters
image_size= x_train.shape[1]
input_shape= (28,28)
batch_size= 128
units = 32
```

```
# create simplernn model
model = Sequential()
model.add(SimpleRNN(units=units,input_shape=input_shape))
model.add(Dense(num_labels))
model.add(Activation('softmax'))
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 simple_rnn_1 (SimpleRNN)    (None, 32)                1952

 dense_1 (Dense)             (None, 10)                330

 activation_1 (Activation)   (None, 10)                0

=================================================================
Total params: 2282 (8.91 KB)
Trainable params: 2282 (8.91 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
# compile the model , define loss function , optimiser , metrics
model.compile(loss='categorical_crossentropy', optimizer='sgd',metrics=['accuracy'])
```

```
# train the model
model.fit(x_train,y_train,epochs=10,batch_size=batch_size)
```

```
Epoch 1/10
469/469 [==============================] - 5s 12ms/step - loss: 0.6316 - accuracy: 0.8162
Epoch 2/10
469/469 [==============================] - 3s 7ms/step - loss: 0.5797 - accuracy: 0.8332
Epoch 3/10
469/469 [==============================] - 3s 7ms/step - loss: 0.5312 - accuracy: 0.8494
Epoch 4/10
```

```
469/469 [==============================] - 4s 8ms/step - loss: 0.5028 - accuracy: 0.8591
Epoch 5/10
469/469 [==============================] - 5s 10ms/step - loss: 0.4820 - accuracy: 0.8654
Epoch 6/10
469/469 [==============================] - 3s 7ms/step - loss: 0.4552 - accuracy: 0.8720
Epoch 7/10
469/469 [==============================] - 3s 7ms/step - loss: 0.4457 - accuracy: 0.8753
Epoch 8/10
469/469 [==============================] - 4s 9ms/step - loss: 0.4285 - accuracy: 0.8803
Epoch 9/10
469/469 [==============================] - 4s 9ms/step - loss: 0.4127 - accuracy: 0.8853
Epoch 10/10
469/469 [==============================] - 3s 7ms/step - loss: 0.3961 - accuracy: 0.8898
<keras.src.callbacks.History at 0x7fea1f73a740>
```

```
# Evaluate the model
loss , acc = model.evaluate(x_test,y_test,batch_size=batch_size)
```

```
79/79 [==============================] - 0s 4ms/step - loss: 0.3772 - accuracy: 0.8943
```

```
print('Test Accuracy :', round(100*acc,2))
```

```
Test Accuracy : 89.43
```

```
#predict
pred = model.predict(x_test[:2])
```

```
1/1 [==============================] - 0s 21ms/step
```

```
# predicted value
pred[1].argmax()
```

```
2
```

```
# actual value
y_test[1].argmax()
```

```
2
```

```
y_test[1]
```

```
array([0., 0., 1., 0., 0., 0., 0., 0., 0., 0.], dtype=float32)
```

```
y_test[0]
```

```
array([0., 0., 0., 0., 0., 0., 0., 1., 0., 0.], dtype=float32)
```

LSTM using MNIST

```
# LSTM model
from keras.layers import LSTM
lstm_model = Sequential()
lstm_model.add(LSTM(units=units,input_shape=input_shape))
lstm_model.add(Dense(num_labels))
lstm_model.add(Activation('softmax'))
lstm_model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_1 (LSTM)               (None, 32)                7808

 dense_1 (Dense)             (None, 10)                330

 activation_1 (Activation)   (None, 10)                0

=================================================================
Total params: 8138 (31.79 KB)
Trainable params: 8138 (31.79 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
# compile the model , define loss function , optimiser , metrics
#lstm_model.compile(loss='categorical_crossentropy', optimizer='sgd',metrics=['accuracy'])
lstm_model.compile(loss='categorical_crossentropy', optimizer='adam',metrics=['accuracy'])
```

```
# train the model
```

```
lstm_model.fit(x_train,y_train,epochs=10,batch_size=batch_size)
```

```
Epoch 1/10
469/469 [==============================] - 11s 19ms/step - loss: 0.3916 - accuracy: 0.8885
Epoch 2/10
469/469 [==============================] - 8s 16ms/step - loss: 0.2707 - accuracy: 0.9216
Epoch 3/10
469/469 [==============================] - 9s 19ms/step - loss: 0.2160 - accuracy: 0.9375
Epoch 4/10
469/469 [==============================] - 9s 19ms/step - loss: 0.1859 - accuracy: 0.9460
Epoch 5/10
469/469 [==============================] - 8s 16ms/step - loss: 0.1609 - accuracy: 0.9524
Epoch 6/10
469/469 [==============================] - 9s 19ms/step - loss: 0.1433 - accuracy: 0.9577
Epoch 7/10
469/469 [==============================] - 7s 15ms/step - loss: 0.1278 - accuracy: 0.9625
Epoch 8/10
469/469 [==============================] - 9s 19ms/step - loss: 0.1170 - accuracy: 0.9649
Epoch 9/10
469/469 [==============================] - 9s 19ms/step - loss: 0.1083 - accuracy: 0.9675
Epoch 10/10
469/469 [==============================] - 8s 17ms/step - loss: 0.0993 - accuracy: 0.9707
<keras.src.callbacks.History at 0x7fe18f600070>
```

```
#predict
lstm_pred = lstm_model.predict(x_test[:2])
```

```
1/1 [==============================] - 1s 510ms/step
```

```
# actual value
y_test[1].argmax()
```

```
2
```

```
# predicted value
lstm_pred[1].argmax()
```

```
2
```

```
y_test[0].argmax()
```

```
7
```

```
lstm_pred[0].argmax()
```

```
7
```