```
# hidden layer parameters
n_cells    =  # number of neurons to add in the hidden layer
time_steps =  # length of sequences
features   =  # number of features of each entity in the sequence

# output layer parameters
n_output          =  # number of classes in case of classification, 1 in case of regression
output_activation =  # "softmax" or "sigmoid" in case of classification, "linear" in case of
regression

============================================================================================
============================
# 1. Vanilla RNN
============================================================================================
============================

# import libraries
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import SimpleRNN


# instantiate the Keras' sequential model
model = Sequential()

# add hidden layer
model.add(SimpleRNN(n_cells, input_shape=(time_steps, features)))

# add output layer
model.add(Dense(n_output, activation=output_activation)

============================================================================================
============================
# 2. Many-to-one RNN
============================================================================================
============================

# import libraries
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import SimpleRNN

# instantiate model
model = Sequential()

# time_steps: multiple input, that is, one input at each timestep
model.add(SimpleRNN(n_cells, input_shape=(time_steps, features)))

# single output at output layer
model.add(Dense(n_classes, activation=output_activation))


============================================================================================
============================
# 3. Many-to-many RNN - input sequence is equal to output sequence
============================================================================================
============================

# import TimeDistributed() layer
from keras.layers import TimeDistributed

# instantiate model
model = Sequential()

# time_steps: multiple input, that is, one input at each timestep
model.add(SimpleRNN(n_cells, input_shape=(time_steps, features)))

# TimeDistributed(): This function is used when you want your neural network to provide an output
```

```
at each timestep which is exactly what we want in the many-to-many RNN model.
model.add(TimeDistributed(Dense(n_classes, activation='softmax')))


================================================================================================
============================
# 4. Encoder-decoder RNN: input sequence is not equal to output sequence
================================================================================================
============================

# import RepeatVector() layer
from keras.layers import RepeatVector

# instantiate model
model = Sequential()

# encoder with multiple inputs
model.add(LSTM(n_cells_input, input_shape=(input_timesteps, ...)))

# encoded sequence
model.add(RepeatVector(output_timesteps))

model.add(LSTM(n_cells_output, return_sequences=True))

# TimeDistributed(): multiple outputs at the output layer
model.add(TimeDistributed(Dense(n_classes, activation='softmax')))


================================================================================================
============================
# 5. One-to-many RNN
================================================================================================
============================

# instantiate model
model = Sequential()

# time_steps is one in this case because the input consists of only one entity
model.add(SimpleRNN(n_cells, input_shape=(1, features)))

# TimeDistributed(): multiple outputs at the output layer
model.add(TimeDistributed(Dense(n_classes, activation='softmax')))

================================================================================================
============================
# 6. Bidirectional RNN
================================================================================================
============================

# import bidirectional layer
from keras.layers import Bidirectional

# instantiate model
model = Sequential()

# bidirectional RNN layer
model.add(Bidirectional(SimpleRNN(n_cells, input_shape=(time_steps, features)))

# output layer
model.add(Dense(n_classes, activation = 'softmax'))


================================================================================================
============================
# 7. LSTM network
================================================================================================
============================

# import LSTM layer
```

```
from keras.layers import LSTM

# instantiate model
model = Sequential()

# replace the SimpleRNN() layer with LSTM() layer
model.add(LSTM(n_cells, input_shape=(time_steps, features)))

# output layer
model.add(Dense(n_classes, activation='softmax'))
```

```
================================================================================================================
============================
# 8. GRU network
================================================================================================================
============================

# import GRU layer
from keras.layers import GRU

# instantiate model
model = Sequential()

# replace the LSTM() layer with GRU() layer
model.add(GRU(n_cells, input_shape=(time_steps, features)))

# output layer
model.add(Dense(n_classes, activation="softmax"))
```