Recurrent Neural Networks

Setup

First, let's make sure this notebook works well in both python 2 and 3, import a few common modules, ensure MatplotLib plots figures inline and prepare a function to save the figures:

In [ ]:
```
# To support both python 2 and python 3
from __future__ import division, print_function, unicode_literals

# Common imports
import numpy as np
import os

# to make this notebook's output stable across runs
def reset_graph(seed=42):
    tf.reset_default_graph()
    tf.set_random_seed(seed)
    np.random.seed(seed)

# To plot pretty figures
%matplotlib inline
import matplotlib
import matplotlib.pyplot as plt
plt.rcParams['axes.labelsize'] = 14
plt.rcParams['xtick.labelsize'] = 12
plt.rcParams['ytick.labelsize'] = 12

# Where to save the figures
PROJECT_ROOT_DIR = "."
CHAPTER_ID = "rnn"

def save_fig(fig_id, tight_layout=True):
    path = os.path.join(PROJECT_ROOT_DIR, "images", CHAPTER_ID, fig_id + ".png")
    print("Saving figure", fig_id)
    if tight_layout:
        plt.tight_layout()
    plt.savefig(path, format='png', dpi=300)
```

In [ ]:
```
import tensorflow.compat.v1 as tf
tf.disable_eager_execution()
# tf.compat.v1.enable_eager_execution()
```

In [ ]:
```
print(tf.__version__)
```

```
2.16.1
```

Basic RNNs

Manual RNN

Use tensorflow to impelment the forward step of RNN. The input size is 3 and the size of state is 5, you need to define the weights and operations for first two steps in tensorflow.

In [ ]:
```
reset_graph()

n_inputs = 3
n_neurons = 5

# TODO : define the place holder for the finput of first time step and the second time step, expect 2 lines of the code
X0 = tf.placeholder(tf.float32, [None, n_inputs])
X1 = tf.placeholder(tf.float32, [None, n_inputs])

# TODO : define the weights of the Basic RNN, don't forget the bias, 3 lines of code exptected
Wx = tf.Variable(tf.random_normal(shape=[n_inputs, n_neurons], dtype=tf.float32))
Wy = tf.Variable(tf.random_normal(shape=[n_neurons, n_neurons], dtype=tf.float32))
b = tf.Variable(tf.zeros([1, n_neurons], dtype=tf.float32))

# TODO : calculate the output for the first time step and second time step
Y0 = tf.tanh(tf.matmul(X0, Wx) + b)
Y1 = tf.tanh(tf.matmul(Y0, Wy) + tf.matmul(X1, Wx) + b)

init = tf.global_variables_initializer()
```

In [ ]:
```
import numpy as np

X0_batch = np.array([[0, 1, 2], [3, 4, 5], [6, 7, 8], [9, 0, 1]]) # t = 0
X1_batch = np.array([[9, 8, 7], [0, 0, 0], [6, 5, 4], [3, 2, 1]]) # t = 1

# TODO : practice to create a session and evaluate on Y0 and Y1, use variable Y0_val to save the output of the first step x
# use Y1_val to save the output of the second step
with tf.Session() as sess:
    tf.global_variables_initializer().run()
    Y0_val, Y1_val = sess.run([Y0, Y1], feed_dict={X0: X0_batch, X1: X1_batch})
```

```
2024-04-05 20:33:56.713206: I tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:388] MLIR V1 optimization pass is not enabl
ed
```

In [ ]:
```
print(Y0_val)
```

```
[[ 0.12020043  0.44310325  0.01446921 -0.8499385   0.99186   ]
 [-0.94592494  0.6544402  -0.99291945 -0.9999941   0.99999976]
 [-0.9987877   0.7968659  -0.9999755  -0.99999976  0.99999976]
 [-0.97810966  0.9999704  -0.5441994  -0.99999976  0.9960072 ]]
```

In [ ]:
```
print(Y1_val)
```

```
[[-0.99999976 -0.90785277 -0.99999976 -0.99999976  0.99999976]
 [ 0.46041262 -0.9532766  -0.9636991   0.9714265   0.1698459 ]
 [-0.9976139  -0.94382584 -0.99999976 -0.9999977   0.99999976]
 [-0.97473055 -0.95060366 -0.9999533  -0.9910696   0.99739045]]
```

Using `static_rnn()`

Note: `tf.contrib.rnn` was partially moved to the core API in TensorFlow 1.2. Most of the `*Cell` and `*Wrapper` classes are now available in `tf.nn.rnn_cell`, and the `tf.contrib.rnn.static_rnn()` function is available as `tf.nn.static_rnn()`.

In [ ]:
```
import tensorflow.keras as keras
```

In [ ]:
```
n_inputs = 3
n_neurons = 5
```

this code is using depreciated tensorflow version: code does not work and tf 2.10 is not installable - earliest version available is 2.13

In [ ]:
```
!pip3 install tensorflow<=2.10
```

```
zsh:1: 2.10 not found
```

In [ ]:
```
!pip3 install tensorflow==2.10
```

```
ERROR: Could not find a version that satisfies the requirement tensorflow==2.10 (from versions: 2.13.0rc0, 2.13.0rc1, 2.13.0rc2,
2.13.0, 2.13.1, 2.14.0rc0, 2.14.0rc1, 2.14.0, 2.14.1, 2.15.0rc0, 2.15.0rc1, 2.15.0, 2.15.1, 2.16.0rc0, 2.16.1)
ERROR: No matching distribution found for tensorflow==2.10
WARNING: You are using pip version 22.0.4; however, version 24.0 is available.
You should consider upgrading via the '/Library/Frameworks/Python.framework/Versions/3.10/bin/python3.10 -m pip install --upgrade
pip' command.
```

In [ ]:
```
reset_graph()
import tensorflow

X0 = tf.placeholder(tf.float32, [None, n_inputs])
X1 = tf.placeholder(tf.float32, [None, n_inputs])
```

```
basic_cell = keras.layers.SimpleRNNCell(units=n_neurons)
output_seqs, states = tf.nn.static_rnn(basic_cell, [X0, X1],
                                       dtype=tf.float32)
Y0, Y1 = output_seqs
```

```
WARNING:tensorflow:From /var/folders/hw/f8pnpzm163q0j3yww182vmgm0000gn/T/ipykernel_24996/1633095047.py:9: static_rnn (from tensorf
low.python.ops.rnn) is deprecated and will be removed in a future version.
Instructions for updating:
Please use `keras.layers.RNN(cell, unroll=True)`, which is equivalent to this API
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[15], line 9
      5 X1 = tf.placeholder(tf.float32, [None, n_inputs])
      8 basic_cell = keras.layers.SimpleRNNCell(units=n_neurons)
----> 9 output_seqs, states = tf.nn.static_rnn(basic_cell, [X0, X1],
     10                                       dtype=tf.float32)
     11 Y0, Y1 = output_seqs

File /Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/site-packages/tensorflow/python/util/deprecation.py:383, in
deprecated.<locals>.deprecated_wrapper.<locals>.new_func(*args, **kwargs)
    375         _PRINTED_WARNING[cls] = True
    376     _log_deprecation(
    377         'From %s: %s (from %s) is deprecated and will be removed %s.\n'
    378         'Instructions for updating:\n%s', _call_location(),
   (...)
    381         'in a future version' if date is None else ('after %s' % date),
    382         instructions)
--> 383 return func(*args, **kwargs)

File /Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/site-packages/tensorflow/python/util/traceback_utils.py:15
3, in filter_traceback.<locals>.error_handler(*args, **kwargs)
    151 except Exception as e:
    152   filtered_tb = _process_traceback_frames(e.__traceback__)
--> 153   raise e.with_traceback(filtered_tb) from None
    154 finally:
    155   del filtered_tb

File /Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/site-packages/tensorflow/python/ops/rnn.py:1432, in static_
rnn(cell, inputs, initial_state, dtype, sequence_length, scope)
   1429   raise ValueError("If no initial_state is provided, argument `dtype` "
   1430                    "must be specified")
   1431 if getattr(cell, "get_initial_state", None) is not None:
-> 1432   state = cell.get_initial_state(
   1433       inputs=None, batch_size=batch_size, dtype=dtype)
   1434 else:
   1435   state = cell.zero_state(batch_size, dtype)

TypeError: SimpleRNNCell.get_initial_state() got an unexpected keyword argument 'inputs'
```

In [ ]:
```
reset_graph()
import tensorflow

X0 = tf.placeholder(tf.float32, [None, n_inputs])
X1 = tf.placeholder(tf.float32, [None, n_inputs])


basic_cell = keras.layers.SimpleRNNCell(units=n_neurons)
output_seqs, states = keras.layers.RNN(basic_cell, [X0, X1],
                                       dtype=tf.float32)
Y0, Y1 = output_seqs
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[38], line 9
      5 X1 = tf.placeholder(tf.float32, [None, n_inputs])
      8 basic_cell = keras.layers.SimpleRNNCell(units=n_neurons)
----> 9 output_seqs, states = keras.layers.RNN(basic_cell, [X0, X1],
     10                                        dtype=tf.float32)
     11 Y0, Y1 = output_seqs

TypeError: cannot unpack non-iterable RNN object
```

In [ ]:
```
init = tf.global_variables_initializer()
```

This code is using depreciated tensorflow version: code does not work and tf 2.10 is not installable - earliest version available is 2.13

In [ ]:
```
X0_batch = np.array([[0, 1, 2], [3, 4, 5], [6, 7, 8], [9, 0, 1]])
X1_batch = np.array([[9, 8, 7], [0, 0, 0], [6, 5, 4], [3, 2, 1]])

# TODO : create a session, initialize the variables and run the graph to get the output of the RNN
with tf.Session() as sess:
    writer = tf.summary.FileWriter("output", sess.graph)
    init.run()
    Y0_val, Y1_val = sess.run([Y0, Y1], feed_dict={X0: X0_batch, X1: X1_batch})
    writer.close()
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[16], line 7
      5 with tf.Session() as sess:
      6     writer = tf.summary.FileWriter("output", sess.graph)
----> 7     init.run()
      8     Y0_val, Y1_val = sess.run([Y0, Y1], feed_dict={X0: X0_batch, X1: X1_batch})
      9     writer.close()

File /Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/site-packages/tensorflow/python/framework/ops.py:1647, in Operation.run(self, feed_dict, session)
   1631 def run(self, feed_dict=None, session=None) -> None:
   1632   """Runs this operation in a `Session`.
   1633
   1634   Calling this method will execute all preceding operations that
   (...)
   1645       none, the default session will be used.
   1646   """
-> 1647   _run_using_default_session(self, feed_dict, self.graph, session)

File /Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/site-packages/tensorflow/python/framework/ops.py:4612, in _run_using_default_session(operation, feed_dict, graph, session)
   4607     raise ValueError("Cannot execute operation using `run()`: No default "
   4608                      "session is registered. Use `with "
   4609                      "sess.as_default():` or pass an explicit session to "
   4610                      "`run(session=sess)`")
   4611   if session.graph is not graph:
-> 4612     raise ValueError("Cannot use the default session to execute operation: "
   4613                      "the operation's graph is different from the "
   4614                      "session's graph. Pass an explicit session to "
   4615                      "run(session=sess).")
   4616 else:
   4617   if session.graph is not graph:

ValueError: Cannot use the default session to execute operation: the operation's graph is different from the session's graph. Pass an explicit session to run(session=sess).
```

In [ ]:
```
Y0_val
```

Out[ ]:
```
array([[ 0.12020043,  0.44310325,  0.01446921, -0.8499385 ,  0.99186   ],
       [-0.94592494,  0.6544402 , -0.99291945, -0.9999941 ,  0.99999976],
       [-0.9987877 ,  0.7968659 , -0.9999755 , -0.99999976,  0.99999976],
       [-0.97810966,  0.9999704 , -0.5441994 , -0.99999976,  0.9960072 ]],
      dtype=float32)
```

In [ ]:
```
Y1_val
```

Out[ ]:
```
array([[-0.99999976, -0.90785277, -0.99999976, -0.99999976,  0.99999976],
       [ 0.46041262, -0.9532766 , -0.9636991 ,  0.9714265 ,  0.1698459 ],
       [-0.9976139 , -0.94382584, -0.99999976, -0.9999977 ,  0.99999976],
       [-0.97473055, -0.95060366, -0.9999533 , -0.9910696 ,  0.99739045]],
      dtype=float32)
```

Packing sequences

In [ ]:
```
n_steps = 2
n_inputs = 3
n_neurons = 5
```

This code is using depreciated tensorflow version: code does not work and tf 2.10 is not installable - earliest version available is 2.13

In [ ]:
```
reset_graph()

X = tf.placeholder(tf.float32, [None, n_steps, n_inputs])
X_seqs = tf.unstack(tf.transpose(X, perm=[1, 0, 2]))

basic_cell = keras.layers.SimpleRNNCell(units=n_neurons)
output_seqs, states = tf.nn.static_rnn(basic_cell, X_seqs,
                                       dtype=tf.float32)
outputs = tf.transpose(tf.stack(output_seqs), perm=[1, 0, 2])
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[20], line 7
      4 X_seqs = tf.unstack(tf.transpose(X, perm=[1, 0, 2]))
      6 basic_cell = keras.layers.SimpleRNNCell(units=n_neurons)
----> 7 output_seqs, states = tf.nn.static_rnn(basic_cell, X_seqs,
      8                                       dtype=tf.float32)
      9 outputs = tf.transpose(tf.stack(output_seqs), perm=[1, 0, 2])

File /Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/site-packages/tensorflow/python/util/deprecation.py:383, in
deprecated.<locals>.deprecated_wrapper.<locals>.new_func(*args, **kwargs)
    375         _PRINTED_WARNING[cls] = True
    376     _log_deprecation(
    377         'From %s: %s (from %s) is deprecated and will be removed %s.\n'
    378         'Instructions for updating:\n%s', _call_location(),
   (...)
    381         'in a future version' if date is None else ('after %s' % date),
    382         instructions)
--> 383 return func(*args, **kwargs)

File /Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/site-packages/tensorflow/python/util/traceback_utils.py:15
3, in filter_traceback.<locals>.error_handler(*args, **kwargs)
    151 except Exception as e:
    152   filtered_tb = _process_traceback_frames(e.__traceback__)
--> 153   raise e.with_traceback(filtered_tb) from None
    154 finally:
    155   del filtered_tb

File /Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/site-packages/tensorflow/python/ops/rnn.py:1432, in static_
rnn(cell, inputs, initial_state, dtype, sequence_length, scope)
   1429   raise ValueError("If no initial_state is provided, argument `dtype` "
   1430                    "must be specified")
   1431 if getattr(cell, "get_initial_state", None) is not None:
-> 1432   state = cell.get_initial_state(
   1433       inputs=None, batch_size=batch_size, dtype=dtype)
   1434 else:
   1435   state = cell.zero_state(batch_size, dtype)

TypeError: SimpleRNNCell.get_initial_state() got an unexpected keyword argument 'inputs'
```

In [ ]:
```
init = tf.global_variables_initializer()
```

In [ ]:
```
X_batch = np.array([
        # t = 0      t = 1
        [[0, 1, 2], [9, 8, 7]], # instance 1
        [[3, 4, 5], [0, 0, 0]], # instance 2
        [[6, 7, 8], [6, 5, 4]], # instance 3
        [[9, 0, 1], [3, 2, 1]], # instance 4
    ])
```

```
with tf.Session() as sess:
    init.run()
    outputs_val = outputs.eval(feed_dict={X: X_batch})
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[22], line 11
      9 with tf.Session() as sess:
     10     init.run()
---> 11     outputs_val = outputs.eval(feed_dict={X: X_batch})

NameError: name 'outputs' is not defined
```

In [ ]:
```
print(outputs_val)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[23], line 1
----> 1 print(outputs_val)

NameError: name 'outputs_val' is not defined
```

In [ ]:
```
print(np.transpose(outputs_val, axes=[1, 0, 2])[1])
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[24], line 1
----> 1 print(np.transpose(outputs_val, axes=[1, 0, 2])[1])

NameError: name 'outputs_val' is not defined
```

Using `dynamic_rnn()`

In [ ]:
```
n_steps = 2
n_inputs = 3
n_neurons = 5
```

This code is using depreciated tensorflow version: code does not work and tf 2.10 is not installable - earliest version available is 2.13

In [ ]:
```
reset_graph()

X = tf.placeholder(tf.float32, [None, n_steps, n_inputs])

basic_cell = keras.layers.SimpleRNNCell(units=n_neurons)
```

```
outputs, states = tf.nn.dynamic_rnn(basic_cell, X, dtype=tf.float32) # TODO : with dynamic_rnn you don't need to transpose and uns
```

```
WARNING:tensorflow:From /var/folders/hw/f8pnpzm163q0j3yww182vmgm0000gn/T/ipykernel_24996/2557284121.py:7: dynamic_rnn (from tensor
flow.python.ops.rnn) is deprecated and will be removed in a future version.
Instructions for updating:
Please use `keras.layers.RNN(cell)`, which is equivalent to this API
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[26], line 7
      3 X = tf.placeholder(tf.float32, [None, n_steps, n_inputs])
      5 basic_cell = keras.layers.SimpleRNNCell(units=n_neurons)
----> 7 outputs, states = tf.nn.dynamic_rnn(basic_cell, X, dtype=tf.float32) # TODO : with dynamic_rnn you don't need to transpose
and unstack your input sequence.

File /Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/site-packages/tensorflow/python/util/deprecation.py:383, in
deprecated.<locals>.deprecated_wrapper.<locals>.new_func(*args, **kwargs)
    375         _PRINTED_WARNING[cls] = True
    376     _log_deprecation(
    377         'From %s: %s (from %s) is deprecated and will be removed %s.\n'
    378         'Instructions for updating:\n%s', _call_location(),
   (...)
    381         'in a future version' if date is None else ('after %s' % date),
    382         instructions)
--> 383 return func(*args, **kwargs)

File /Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/site-packages/tensorflow/python/util/traceback_utils.py:15
3, in filter_traceback.<locals>.error_handler(*args, **kwargs)
    151 except Exception as e:
    152   filtered_tb = _process_traceback_frames(e.__traceback__)
--> 153   raise e.with_traceback(filtered_tb) from None
    154 finally:
    155   del filtered_tb

File /Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/site-packages/tensorflow/python/ops/rnn.py:727, in dynamic_
rnn(cell, inputs, sequence_length, initial_state, dtype, parallel_iterations, swap_memory, time_major, scope)
    724   raise ValueError("If no initial_state is provided, argument `dtype` "
    725                    "must be specified")
    726 if getattr(cell, "get_initial_state", None) is not None:
--> 727   state = cell.get_initial_state(
    728       inputs=None, batch_size=batch_size, dtype=dtype)
    729 else:
    730   state = cell.zero_state(batch_size, dtype)

TypeError: SimpleRNNCell.get_initial_state() got an unexpected keyword argument 'inputs'
```

In [ ]:
```
init = tf.global_variables_initializer()
```

In [ ]:
```
X_batch = np.array([
        [[0, 1, 2], [9, 8, 7]], # instance 1
```

```
        [[3, 4, 5], [0, 0, 0]], # instance 2
        [[6, 7, 8], [6, 5, 4]], # instance 3
        [[9, 0, 1], [3, 2, 1]], # instance 4
    ])

with tf.Session() as sess:
    init.run()
    outputs_val = outputs.eval(feed_dict={X: X_batch})
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[28], line 10
      8 with tf.Session() as sess:
      9     init.run()
---> 10     outputs_val = outputs.eval(feed_dict={X: X_batch})

NameError: name 'outputs' is not defined
```

In [ ]:
```
print(outputs_val)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[29], line 1
----> 1 print(outputs_val)

NameError: name 'outputs_val' is not defined
```

Setting the sequence lengths

In [ ]:
```
n_steps = 2
n_inputs = 3
n_neurons = 5

reset_graph()

X = tf.placeholder(tf.float32, [None, n_steps, n_inputs])
basic_cell = keras.layers.SimpleRNNCell(units=n_neurons)
```

This code is using depreciated tensorflow version: code does not work and tf 2.10 is not installable - earliest version available is 2.13

In [ ]:
```
# TODO : create a placeholder for seq_length and set this as an extra parameter in dynamic_rnn, expect 2 lines of code
seq_length = tf.placeholder(tf.int32, [None])
outputs, states = tf.nn.dynamic_rnn(basic_cell, X, dtype=tf.float32, sequence_length=seq_length)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[31], line 3
      1 # TODO : create a placeholder for seq_length and set this as an extra parameter in dynamic_rnn, expect 2 lines of code
      2 seq_length = tf.placeholder(tf.int32, [None])
----> 3 outputs, states = tf.nn.dynamic_rnn(basic_cell, X, dtype=tf.float32, sequence_length=seq_length)

File /Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/site-packages/tensorflow/python/util/deprecation.py:383, in deprecated.<locals>.deprecated_wrapper.<locals>.new_func(*args, **kwargs)
    375         _PRINTED_WARNING[cls] = True
    376     _log_deprecation(
    377         'From %s: %s (from %s) is deprecated and will be removed %s.\n'
    378         'Instructions for updating:\n%s', _call_location(),
   (...)
    381         'in a future version' if date is None else ('after %s' % date),
    382         instructions)
--> 383 return func(*args, **kwargs)

File /Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/site-packages/tensorflow/python/util/traceback_utils.py:153, in filter_traceback.<locals>.error_handler(*args, **kwargs)
    151 except Exception as e:
    152   filtered_tb = _process_traceback_frames(e.__traceback__)
--> 153   raise e.with_traceback(filtered_tb) from None
    154 finally:
    155   del filtered_tb

File /Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/site-packages/tensorflow/python/ops/rnn.py:727, in dynamic_rnn(cell, inputs, sequence_length, initial_state, dtype, parallel_iterations, swap_memory, time_major, scope)
    724   raise ValueError("If no initial_state is provided, argument `dtype` "
    725                    "must be specified")
    726 if getattr(cell, "get_initial_state", None) is not None:
--> 727   state = cell.get_initial_state(
    728       inputs=None, batch_size=batch_size, dtype=dtype)
    729 else:
    730   state = cell.zero_state(batch_size, dtype)

TypeError: SimpleRNNCell.get_initial_state() got an unexpected keyword argument 'inputs'
```

In [ ]:
```
init = tf.global_variables_initializer()
```

In [ ]:
```
X_batch = np.array([
        # step 0     step 1
        [[0, 1, 2], [9, 8, 7]], # instance 1
        [[3, 4, 5], [0, 0, 0]], # instance 2 (padded with zero vectors)
        [[6, 7, 8], [6, 5, 4]], # instance 3
        [[9, 0, 1], [3, 2, 1]], # instance 4
    ])
seq_length_batch = np.array([2, 1, 2, 2])
```

In [ ]:
```
with tf.Session() as sess:
    init.run()
    outputs_val, states_val = sess.run(
        [outputs, states], feed_dict={X: X_batch, seq_length: seq_length_batch})
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[34], line 4
      1 with tf.Session() as sess:
      2     init.run()
      3     outputs_val, states_val = sess.run(
----> 4         [outputs, states], feed_dict={X: X_batch, seq_length: seq_length_batch})

NameError: name 'outputs' is not defined
```

In [ ]:
```
print(outputs_val)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[35], line 1
----> 1 print(outputs_val)

NameError: name 'outputs_val' is not defined
```

In [ ]:
```
print(states_val)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[36], line 1
----> 1 print(states_val)

NameError: name 'states_val' is not defined
```

Training a sequence classifier

In [ ]:
```
reset_graph()

n_steps = 28
n_inputs = 28
n_neurons = 150
n_outputs = 10

learning_rate = 0.001

X = tf.placeholder(tf.float32, [None, n_steps, n_inputs])
y = tf.placeholder(tf.int32, [None])

basic_cell = tf.nn.rnn_cell.BasicRNNCell(num_units=n_neurons) # TODO : define a basic cell with BasicRNNCell
```

```
outputs, states = tf.nn.dynamic_rnn(basic_cell, X, dtype=tf.float32) # TODO : create a rnn model with cell as a basic build block

logits = tf.layers.dense(states, n_outputs) # TODO : create a dense layer
xentropy = tf.nn.sparse_softmax_cross_entropy_with_logits(labels=y, logits=logits) # TODO : create loss function with sparse_softm
loss = tf.reduce_mean(xentropy) # TODO : call tf.reduce_mean on the loss you defined
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate) # TODO : define an AdamOptimizer with learning rate equals to lear
training_op = optimizer.minimize(loss) # TODO : minimize the loss
correct = tf.nn.in_top_k(logits, y, 1) # TODO : count the number of correct prediction, you can use tf.nn.in_top_k
accuracy = tf.reduce_mean(tf.cast(correct, tf.float32)) # calcuate the accuracy with tf.reduce_mean

init = tf.global_variables_initializer()
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[37], line 13
     10 X = tf.placeholder(tf.float32, [None, n_steps, n_inputs])
     11 y = tf.placeholder(tf.int32, [None])
---> 13 basic_cell = tf.nn.rnn_cell.BasicRNNCell(num_units=n_neurons) # TODO : define a basic cell with BasicRNNCell
     14 outputs, states = tf.nn.dynamic_rnn(basic_cell, X, dtype=tf.float32) # TODO : create a rnn model with cell as a basic buil
d block with dynamic_rnn
     16 logits = tf.layers.dense(states, n_outputs) # TODO : create a dense layer

File /Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/site-packages/tensorflow/python/util/lazy_loader.py:207, in
KerasLazyLoader.__getattr__(self, item)
    200     raise AttributeError(
    201         "`tf.compat.v2.keras` is not available with Keras 3. Just use "
    202         "`import keras` instead."
    203     )
    204   elif self._tfll_submodule and self._tfll_submodule.startswith(
    205       "__internal__.legacy."
    206   ):
--> 207     raise AttributeError(
    208         f"`{item}` is not available with Keras 3."
    209     )
    210 module = self._load()
    211 return getattr(module, item)

AttributeError: `BasicRNNCell` is not available with Keras 3.
```

In [ ]:
```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import SimpleRNN, Dense

n_steps = 28
n_inputs = 28
n_neurons = 150
n_outputs = 10
learning_rate = 0.001

# Building the model using the Keras Sequential API
```

```
model = Sequential([
    # RNN layer
    SimpleRNN(n_neurons, input_shape=(n_steps, n_inputs)),
    # Dense layer for output
    Dense(n_outputs, activation='softmax')
])

# Compile the model
model.compile(loss='sparse_categorical_crossentropy',
              optimizer=tf.keras.optimizers.Adam(learning_rate=learning_rate),
              metrics=['accuracy'])

model.summary()
```

```
ERROR:tensorflow:==================================
Object was never used (type <class 'tensorflow.python.framework.ops.Operation'>):
<tf.Operation 'sequential/simple_rnn/simple_rnn_cell/Assert/Assert' type=Assert>
If you want to mark it as used call its "mark_used()" method.
It was originally created here:
  File "/Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/site-packages/tensorflow/python/ops/weak_tensor_ops.py",
line 88, in wrapper
    return op(*args, **kwargs)  File "/Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/site-packages/tensorflow/p
ython/ops/numpy_ops/np_array_ops.py", line 316, in diag
    control_flow_assert.Assert(  File "/Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/site-packages/tensorflow/
python/util/traceback_utils.py", line 155, in error_handler
    del filtered_tb  File "/Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/site-packages/tensorflow/python/util/
dispatch.py", line 1260, in op_dispatch_handler
    return dispatch_target(*args, **kwargs)  File "/Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/site-package
s/tensorflow/python/util/tf_should_use.py", line 288, in wrapped
    return _add_should_use_warning(fn(*args, **kwargs),
==================================
```

```
/Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/site-packages/keras/src/layers/rnn/rnn.py:204: UserWarning: Do n
ot pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as t
he first layer in the model instead.
  super().__init__(**kwargs)
```

Model: "sequential"

Layer (type)	Output Shape	Param #
simple_rnn (SimpleRNN)	(None, 150)	26,850
dense (Dense)	(None, 10)	1,510

Total params: 28,360 (110.78 KB)

Trainable params: 28,360 (110.78 KB)

Non-trainable params: 0 (0.00 B)

Warning: `tf.examples.tutorials.mnist` is deprecated. We will use `tf.keras.datasets.mnist` instead.

In [ ]:
```
(X_train, y_train), (X_test, y_test) = tf.keras.datasets.mnist.load_data()
X_train = X_train.astype(np.float32).reshape(-1, 28*28) / 255.0
X_test = X_test.astype(np.float32).reshape(-1, 28*28) / 255.0
y_train = y_train.astype(np.int32)
y_test = y_test.astype(np.int32)
X_valid, X_train = X_train[:5000], X_train[5000:]
y_valid, y_train = y_train[:5000], y_train[5000:]
```

In [ ]:
```
def shuffle_batch(X, y, batch_size):
    rnd_idx = np.random.permutation(len(X))
    n_batches = len(X) // batch_size
    for batch_idx in np.array_split(rnd_idx, n_batches):
        X_batch, y_batch = X[batch_idx], y[batch_idx]
        yield X_batch, y_batch
```

In [ ]:
```
X_test = X_test.reshape((-1, n_steps, n_inputs))
```

In [ ]:
```
n_epochs = 100
batch_size = 150

with tf.Session() as sess:
    init.run()
    for epoch in range(n_epochs):
        for X_batch, y_batch in shuffle_batch(X_train, y_train, batch_size):
            X_batch = X_batch.reshape((-1, n_steps, n_inputs))
            sess.run(training_op, feed_dict={X: X_batch, y: y_batch})
        acc_batch = accuracy.eval(feed_dict={X: X_batch, y: y_batch})
        acc_test = accuracy.eval(feed_dict={X: X_test, y: y_test})
        print(epoch, "Last batch accuracy:", acc_batch, "Test accuracy:", acc_test)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[42], line 4
      1 n_epochs = 100
      2 batch_size = 150
----> 4 with tf.Session() as sess:
      5     init.run()
      6     for epoch in range(n_epochs):

AttributeError: module 'tensorflow' has no attribute 'Session'
```

In [ ]:
```
print("Eager execution:", tf.executing_eagerly())
print(tf.__version__)
```

```
Eager execution: False
2.16.1
```

In [ ]:
```
import numpy as np
from tensorflow.keras.utils import to_categorical
```

```
tf.compat.v1.enable_eager_execution()

# Assuming X_train, y_train, X_test, y_test are already defined and preprocessed
n_epochs = 100
batch_size = 150

# Reshape the input data to match the expected input of the RNN
X_train_reshaped = X_train.reshape((-1, n_steps, n_inputs))
X_test_reshaped = X_test.reshape((-1, n_steps, n_inputs))

# Convert labels to one-hot encoding if they are not already
y_train_categorical = to_categorical(y_train, num_classes=n_outputs)
y_test_categorical = to_categorical(y_test, num_classes=n_outputs)

# Train the model
history = model.fit(X_train_reshaped, y_train_categorical, epochs=n_epochs,
                    batch_size=batch_size, validation_data=(X_test_reshaped, y_test_categorical))

# Evaluate the model
evaluation = model.evaluate(X_test_reshaped, y_test_categorical)
print('Test loss:', evaluation[0])
print('Test accuracy:', evaluation[1])
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[72], line 3
      1 import numpy as np
      2 from tensorflow.keras.utils import to_categorical
----> 3 tf.compat.v1.enable_eager_execution()
      5 # Assuming X_train, y_train, X_test, y_test are already defined and preprocessed
      6 n_epochs = 100

File /Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/site-packages/tensorflow/python/framework/ops.py:4944, in enable_eager_execution(config, device_policy, execution_mode)
   4942 logging.vlog(1, "Enabling eager execution")
   4943 if context.default_execution_mode != context.EAGER_MODE:
-> 4944   return enable_eager_execution_internal(
   4945       config=config,
   4946       device_policy=device_policy,
   4947       execution_mode=execution_mode,
   4948       server_def=None)

File /Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/site-packages/tensorflow/python/framework/ops.py:5008, in enable_eager_execution_internal(config, device_policy, execution_mode, server_def)
   5005   graph_mode_has_been_used = (
   5006       _default_graph_stack._global_default_graph is not None)  # pylint: disable=protected-access
   5007   if graph_mode_has_been_used:
-> 5008     raise ValueError(
   5009         "tf.enable_eager_execution must be called at program startup.")
   5010 context.default_execution_mode = context.EAGER_MODE
   5011 # pylint: disable=protected-access

ValueError: tf.enable_eager_execution must be called at program startup.
```

Multi-layer RNN

In [ ]:
```
reset_graph()

n_steps = 28
n_inputs = 28
n_outputs = 10

learning_rate = 0.001

X = tf.placeholder(tf.float32, [None, n_steps, n_inputs])
y = tf.placeholder(tf.int32, [None])
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[44], line 1
----> 1 reset_graph()
      3 n_steps = 28
      4 n_inputs = 28

Cell In[1], line 10, in reset_graph(seed)
      9 def reset_graph(seed=42):
---> 10     tf.reset_default_graph()
     11     tf.set_random_seed(seed)
     12     np.random.seed(seed)

AttributeError: module 'tensorflow' has no attribute 'reset_default_graph'
```

In [ ]:
```
n_neurons = 100
n_layers = 3


layers = # TODO : build a cell list that contains multiple cells, you can use list comprehension to generate cell list.

multi_layer_cell =  # TODO : build a MultiRNNCell with list of BasicRNNCell objects (Hint: Read StackedRNNCells documentation)

outputs, states =  # TODO : put the multi_layer_cell in dynamic_rnn
```

In [ ]:
```
states_concat = tf.concat(axis=1, values=states)
logits = tf.layers.dense(states_concat, n_outputs)
xentropy = tf.nn.sparse_softmax_cross_entropy_with_logits(labels=y, logits=logits)
loss = tf.reduce_mean(xentropy)
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate)
training_op = optimizer.minimize(loss)
correct = tf.nn.in_top_k(logits, y, 1)
accuracy = tf.reduce_mean(tf.cast(correct, tf.float32))

init = tf.global_variables_initializer()
```

In [ ]:
```
n_epochs = 10
batch_size = 150

with tf.Session() as sess:
    init.run()
    for epoch in range(n_epochs):
        for X_batch, y_batch in shuffle_batch(X_train, y_train, batch_size):
            X_batch = X_batch.reshape((-1, n_steps, n_inputs))
            sess.run(training_op, feed_dict={X: X_batch, y: y_batch})
        acc_batch = accuracy.eval(feed_dict={X: X_batch, y: y_batch})
        acc_test = accuracy.eval(feed_dict={X: X_test, y: y_test})
        print(epoch, "Last batch accuracy:", acc_batch, "Test accuracy:", acc_test)
```

Time series

In [ ]:
```
t_min, t_max = 0, 30
resolution = 0.1

def time_series(t):
    return t * np.sin(t) / 3 + 2 * np.sin(t*5)

def next_batch(batch_size, n_steps):
    t0 = np.random.rand(batch_size, 1) * (t_max - t_min - n_steps * resolution)
    Ts = t0 + np.arange(0., n_steps + 1) * resolution
    ys = time_series(Ts)
    return ys[:, :-1].reshape(-1, n_steps, 1), ys[:, 1:].reshape(-1, n_steps, 1)
```

In [ ]:
```
t = np.linspace(t_min, t_max, int((t_max - t_min) / resolution))

n_steps = 20
t_instance = np.linspace(12.2, 12.2 + resolution * (n_steps + 1), n_steps + 1)

plt.figure(figsize=(11,4))
plt.subplot(121)
plt.title("A time series (generated)", fontsize=14)
plt.plot(t, time_series(t), label=r"$t . \sin(t) / 3 + 2 . \sin(5t)$")
plt.plot(t_instance[:-1], time_series(t_instance[:-1]), "b-", linewidth=3, label="A training instance")
plt.legend(loc="lower left", fontsize=14)
plt.axis([0, 30, -17, 13])
plt.xlabel("Time")
plt.ylabel("Value")

plt.subplot(122)
plt.title("A training instance", fontsize=14)
plt.plot(t_instance[:-1], time_series(t_instance[:-1]), "bo", markersize=10, label="instance")
plt.plot(t_instance[1:], time_series(t_instance[1:]), "w*", markersize=10, label="target")
plt.legend(loc="upper left")
plt.xlabel("Time")


save_fig("time_series_plot")
plt.show()
```

In [ ]:
```
X_batch, y_batch = next_batch(1, n_steps)
```

In [ ]:
```
np.c_[X_batch[0], y_batch[0]]
```

Without using an `OutputProjectionWrapper`

Now let's create the RNN. It will contain 100 recurrent neurons and we will unroll it over 20 time steps since each training instance will be 20 inputs long. Each input will contain only one feature (the value at that time). The targets are also sequences of 20 inputs, each containing a single value:

In [ ]:
```
reset_graph()

n_steps = 20
n_inputs = 1
n_neurons = 100
n_outputs = 1

X = tf.placeholder(tf.float32, [None, n_steps, n_inputs])
y = tf.placeholder(tf.float32, [None, n_steps, n_outputs])
```

In [ ]:
```
cell = keras.layers.SimpleRNNCell(units=n_neurons, activation=tf.nn.relu)
rnn_outputs, states = tf.nn.dynamic_rnn(cell, X, dtype=tf.float32)
```

In [ ]:
```
learning_rate = 0.001
```

In [ ]:
```
stacked_rnn_outputs = tf.reshape(rnn_outputs, [-1, n_neurons])
stacked_outputs = tf.layers.dense(stacked_rnn_outputs, n_outputs)
outputs = tf.reshape(stacked_outputs, [-1, n_steps, n_outputs])
```

In [ ]:
```
loss = tf.reduce_mean(tf.square(outputs - y))
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate)
training_op = optimizer.minimize(loss)

init = tf.global_variables_initializer()
saver = tf.train.Saver()
```

In [ ]:
```
n_iterations = 1500
batch_size = 50

with tf.Session() as sess:
    init.run()
    for iteration in range(n_iterations):
        X_batch, y_batch = next_batch(batch_size, n_steps)
        sess.run(training_op, feed_dict={X: X_batch, y: y_batch})
        if iteration % 100 == 0:
            mse = loss.eval(feed_dict={X: X_batch, y: y_batch})
            print(iteration, "\tMSE:", mse)

    X_new = time_series(np.array(t_instance[:-1].reshape(-1, n_steps, n_inputs)))
    y_pred = sess.run(outputs, feed_dict={X: X_new})

    saver.save(sess, "./my_time_series_model")
```

In [ ]:
```
y_pred
```

In [ ]:
```
plt.title("Testing the model", fontsize=14)
plt.plot(t_instance[:-1], time_series(t_instance[:-1]), "bo", markersize=10, label="instance")
plt.plot(t_instance[1:], time_series(t_instance[1:]), "w*", markersize=10, label="target")
plt.plot(t_instance[1:], y_pred[0,:,0], "r.", markersize=10, label="prediction")
plt.legend(loc="upper left")
plt.xlabel("Time")

plt.show()
```

Generating a creative new sequence

In [ ]:
```
with tf.Session() as sess:                          # not shown in the book
    saver.restore(sess, "./my_time_series_model") # not shown

    sequence = [0.] * n_steps
    for iteration in range(300):
        X_batch = np.array(sequence[-n_steps:]).reshape(1, n_steps, 1)
        y_pred = sess.run(outputs, feed_dict={X: X_batch})
        sequence.append(y_pred[0, -1, 0])
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[45], line 1
----> 1 with tf.Session() as sess:                          # not shown in the book
      2     saver.restore(sess, "./my_time_series_model") # not shown
      4     sequence = [0.] * n_steps

AttributeError: module 'tensorflow' has no attribute 'Session'
```

In [ ]:
```
plt.figure(figsize=(8,4))
plt.plot(np.arange(len(sequence)), sequence, "b-")
plt.plot(t[:n_steps], sequence[:n_steps], "b-", linewidth=3)
plt.xlabel("Time")
plt.ylabel("Value")
plt.show()
```

In [ ]:
```
with tf.Session() as sess:
    saver.restore(sess, "./my_time_series_model")

    sequence1 = [0. for i in range(n_steps)]
    for iteration in range(len(t) - n_steps):
        X_batch = np.array(sequence1[-n_steps:]).reshape(1, n_steps, 1)
        y_pred = sess.run(outputs, feed_dict={X: X_batch})
        sequence1.append(y_pred[0, -1, 0])
```

```
    sequence2 = [time_series(i * resolution + t_min + (t_max-t_min/3)) for i in range(n_steps)]
    for iteration in range(len(t) - n_steps):
        X_batch = np.array(sequence2[-n_steps:]).reshape(1, n_steps, 1)
        y_pred = sess.run(outputs, feed_dict={X: X_batch})
        sequence2.append(y_pred[0, -1, 0])

plt.figure(figsize=(11,4))
plt.subplot(121)
plt.plot(t, sequence1, "b-")
plt.plot(t[:n_steps], sequence1[:n_steps], "b-", linewidth=3)
plt.xlabel("Time")
plt.ylabel("Value")

plt.subplot(122)
plt.plot(t, sequence2, "b-")
plt.plot(t[:n_steps], sequence2[:n_steps], "b-", linewidth=3)
plt.xlabel("Time")
save_fig("creative_sequence_plot")
plt.show()
```

Train a RNN model on external dataset

Now you've trained a RNN model on the data generated by a simple function, you've practiced how to deal with time series data. It's your turn to apply it on your favoriate time series dataset, the potential datasets can be found here https://archive.ics.uci.edu/ml/datasets.php?format=&task=&att=&area=&numAtt=10to100&numIns=&type=ts&sort=nameUp&view=table.

You have to build a LSTM model, it's almost the same as defining a basic rnn cell we've covered with tf.contrib.rnn.BasicLSTMCell, you may want to use tf.contrib.rnn.BasicLSTMCell to build your cell. And you need to define the appropriate input, output and the hidden state size. Build a graph and train on a batch of data. Finally you need to create a session to execute the graph and evaluate your model. You can replace the LSTM cell with RNN cell and tune the size of hidden state to see how they influence the model's performance.

Your code should be clean and organized, you need to elaborate how different models and hyperparameter influence your RNN's performance.

In [ ]:
```
# import libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import tensorflow as tf

import os
```

In [ ]:
```
# importing training data
df_train = pd.read_csv('data/Google_Stock_Price_Train.csv')
```

```
# remove dates and set as np array
train_array = df_train.iloc[:, 1:2].values
```

In [ ]:
```
# scale features - minmax for normalization
from sklearn.preprocessing import MinMaxScaler

# initialize scalar
sc = MinMaxScaler()
train_array_normalized = sc.fit_transform(train_array)
```

In [ ]:
```
# create data structures
x_train = []
y_train =  []

# recalling 60 previous stock prices for x_train to predict y_train
for i in range(60, len(train_array_normalized)):

    # creates a sliding window
    x_train.append(train_array_normalized[i-60:i, 0])
    y_train.append(train_array_normalized[i, 0])

# convert training lists to np.array
x_train = np.array(x_train)
y_train = np.array(y_train)
```

In [ ]:
```
# reshape data structures - num of predictors
x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1], 1))
```

In [ ]:
```
# building RNN architecture
# !pip3 install tensorflow
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers import Dropout

# initilising RNN
regressor = Sequential()
```

In [ ]:
```
# create first LSTM layer and dropout regularisation
regressor.add(LSTM(
    units=50, # number of neurons for this LSTM layer
    return_sequences=True, # becuase we're going to make a stacked LSTM network
    input_shape=(x_train.shape[1], 1) # only contains the last two dimensions of x_train
))

regressor.add(Dropout(
    rate=0.2 # 20% of the neurons will be ignored/dropped out during training
))
```

```
/Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/site-packages/keras/src/layers/rnn/rnn.py:204: UserWarning: Do n
ot pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as t
he first layer in the model instead.
  super().__init__(**kwargs)
```

In [ ]:
```
# create second LSTM layer and dropout regularisation
regressor.add(LSTM(
    units=50, # number of neurons for this LSTM layer - keeping higher dimensionality
    return_sequences=True # becuase we're going to make a stacked LSTM network
))

regressor.add(Dropout(
    rate=0.2 # 20% of the neurons will be ignored/dropped out during training
))
```

In [ ]:
```
# create third LSTM layer and dropout regularisation
regressor.add(LSTM(
    units=50, # number of neurons for this LSTM layer - keeping higher dimensionality
    return_sequences=True # becuase we're going to make a stacked LSTM network
))

regressor.add(Dropout(
    rate=0.2 # 20% of the neurons will be ignored/dropped out during training
))
```

In [ ]:
```
# create fourth (last) LSTM layer and dropout regularisation
regressor.add(LSTM(
    units=50, # number of neurons for this LSTM layer - keeping higher dimensionality
    return_sequences=False # becuase this is our final layer
))

regressor.add(Dropout(
    rate=0.2 # 20% of the neurons will be ignored/dropped out during training
))
```

In [ ]:
```
# output layer
regressor.add(
    Dense(
        units=1 # the number of units/dimension in the output - stock price
    )
)
```

In [ ]:
```
# compile the RNN with optimizer
regressor.compile(
    optimizer='adam',
    loss='mean_squared_error'
)
```

In [ ]:
```
# fit RNN to training set
regressor.fit(
    x_train,
    y_train,
    epochs=100, # number of iterations
    batch_size=32 # batches of stock prices to back and forward propogate on
)
```

```
Epoch 1/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 3s 39ms/step - loss: 0.1230
Epoch 2/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 1s 39ms/step - loss: 0.0071
Epoch 3/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 2s 44ms/step - loss: 0.0052
Epoch 4/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 2s 42ms/step - loss: 0.0052
Epoch 5/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 2s 42ms/step - loss: 0.0054
Epoch 6/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 2s 41ms/step - loss: 0.0047
Epoch 7/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 2s 41ms/step - loss: 0.0051
Epoch 8/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 2s 42ms/step - loss: 0.0046
Epoch 9/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 2s 41ms/step - loss: 0.0044
Epoch 10/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 2s 40ms/step - loss: 0.0042
Epoch 11/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 2s 41ms/step - loss: 0.0053
Epoch 12/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 2s 40ms/step - loss: 0.0049
Epoch 13/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 2s 40ms/step - loss: 0.0047
Epoch 14/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 2s 40ms/step - loss: 0.0040
Epoch 15/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 2s 40ms/step - loss: 0.0036
Epoch 16/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 2s 40ms/step - loss: 0.0037
Epoch 17/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 2s 42ms/step - loss: 0.0045
Epoch 18/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 2s 40ms/step - loss: 0.0037
Epoch 19/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 2s 40ms/step - loss: 0.0031
Epoch 20/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 2s 40ms/step - loss: 0.0041
Epoch 21/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 2s 40ms/step - loss: 0.0036
Epoch 22/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 2s 41ms/step - loss: 0.0035
Epoch 23/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 2s 41ms/step - loss: 0.0036
Epoch 24/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 2s 40ms/step - loss: 0.0030
Epoch 25/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 2s 40ms/step - loss: 0.0032
```

```
Epoch 26/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 2s 40ms/step - loss: 0.0032
Epoch 27/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 2s 42ms/step - loss: 0.0032
Epoch 28/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 2s 40ms/step - loss: 0.0032
Epoch 29/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 2s 40ms/step - loss: 0.0033
Epoch 30/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 2s 41ms/step - loss: 0.0028
Epoch 31/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 2s 40ms/step - loss: 0.0025
Epoch 32/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 2s 41ms/step - loss: 0.0030
Epoch 33/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 2s 41ms/step - loss: 0.0032
Epoch 34/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 2s 40ms/step - loss: 0.0033
Epoch 35/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 2s 41ms/step - loss: 0.0033
Epoch 36/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 2s 41ms/step - loss: 0.0029
Epoch 37/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 2s 40ms/step - loss: 0.0026
Epoch 38/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 2s 41ms/step - loss: 0.0026
Epoch 39/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 2s 41ms/step - loss: 0.0028
Epoch 40/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 2s 40ms/step - loss: 0.0028
Epoch 41/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 2s 41ms/step - loss: 0.0030
Epoch 42/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 2s 41ms/step - loss: 0.0025
Epoch 43/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 2s 41ms/step - loss: 0.0025
Epoch 44/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 2s 40ms/step - loss: 0.0026
Epoch 45/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 2s 40ms/step - loss: 0.0023
Epoch 46/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 2s 42ms/step - loss: 0.0025
Epoch 47/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 2s 43ms/step - loss: 0.0026
Epoch 48/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 2s 41ms/step - loss: 0.0021
Epoch 49/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 2s 41ms/step - loss: 0.0026
Epoch 50/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 2s 40ms/step - loss: 0.0024
```

```
Epoch 51/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 2s 42ms/step - loss: 0.0024
Epoch 52/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 2s 42ms/step - loss: 0.0025
Epoch 53/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 2s 40ms/step - loss: 0.0020
Epoch 54/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 2s 40ms/step - loss: 0.0024
Epoch 55/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 2s 40ms/step - loss: 0.0024
Epoch 56/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 2s 42ms/step - loss: 0.0025
Epoch 57/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 2s 41ms/step - loss: 0.0025
Epoch 58/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 2s 40ms/step - loss: 0.0029
Epoch 59/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 2s 40ms/step - loss: 0.0020
Epoch 60/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 2s 41ms/step - loss: 0.0024
Epoch 61/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 2s 46ms/step - loss: 0.0020
Epoch 62/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 2s 41ms/step - loss: 0.0022
Epoch 63/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 2s 40ms/step - loss: 0.0019
Epoch 64/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 2s 40ms/step - loss: 0.0023
Epoch 65/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 2s 41ms/step - loss: 0.0022
Epoch 66/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 2s 41ms/step - loss: 0.0021
Epoch 67/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 2s 41ms/step - loss: 0.0021
Epoch 68/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 2s 40ms/step - loss: 0.0020
Epoch 69/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 2s 41ms/step - loss: 0.0017
Epoch 70/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 2s 41ms/step - loss: 0.0020
Epoch 71/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 2s 42ms/step - loss: 0.0020
Epoch 72/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 2s 40ms/step - loss: 0.0022
Epoch 73/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 2s 40ms/step - loss: 0.0018
Epoch 74/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 2s 41ms/step - loss: 0.0018
Epoch 75/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 2s 41ms/step - loss: 0.0019
```

```
Epoch 76/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 2s 40ms/step - loss: 0.0019
Epoch 77/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 2s 41ms/step - loss: 0.0020
Epoch 78/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 2s 40ms/step - loss: 0.0016
Epoch 79/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 2s 40ms/step - loss: 0.0020
Epoch 80/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 2s 40ms/step - loss: 0.0017
Epoch 81/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 2s 40ms/step - loss: 0.0017
Epoch 82/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 2s 40ms/step - loss: 0.0019
Epoch 83/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 2s 41ms/step - loss: 0.0016
Epoch 84/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 2s 41ms/step - loss: 0.0015
Epoch 85/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 2s 41ms/step - loss: 0.0017
Epoch 86/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 2s 40ms/step - loss: 0.0015
Epoch 87/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 2s 40ms/step - loss: 0.0017
Epoch 88/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 2s 41ms/step - loss: 0.0019
Epoch 89/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 2s 40ms/step - loss: 0.0015
Epoch 90/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 2s 42ms/step - loss: 0.0016
Epoch 91/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 2s 41ms/step - loss: 0.0015
Epoch 92/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 2s 40ms/step - loss: 0.0016
Epoch 93/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 2s 41ms/step - loss: 0.0014
Epoch 94/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 2s 42ms/step - loss: 0.0017
Epoch 95/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 2s 40ms/step - loss: 0.0017
Epoch 96/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 2s 41ms/step - loss: 0.0016
Epoch 97/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 2s 40ms/step - loss: 0.0012
Epoch 98/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 2s 40ms/step - loss: 0.0014
Epoch 99/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 2s 41ms/step - loss: 0.0014
Epoch 100/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 2s 42ms/step - loss: 0.0015
```

Out[ ]: <keras.src.callbacks.history.History at 0x28a03afe0>

In [ ]:
```
# get test set
df_test = pd.read_csv('data/Google_Stock_Price_Test.csv')

# remove dates, isolate close price and set as np array
y_test_array = df_test.iloc[:, 1:2].values
```

In [ ]:
```
# predict price
df_all = pd.concat(
    (
        df_train['Open'],
        df_test['Open']
    ),
    axis=0
)

# input to predict from - predicting the next 60 stock prices
inputs = df_all[len(df_all) - len(df_test) - 60:].values
inputs = inputs.reshape(-1, 1)
inputs = sc.transform(inputs)

x_test = []
# recalling 60 previous stock prices for x_train to predict y_train
for i in range(60, 80): # the test set only contains 20 days to predict on, 60+20 = 80

    # creates a sliding window
    x_test.append(inputs[i-60:i, 0])

x_test = np.array(x_test)
x_test = np.reshape(x_test, (x_test.shape[0], x_test.shape[1], 1))

y_pred_array = regressor.predict(x_test)
y_pred_array = sc.inverse_transform(y_pred_array)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 205ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 205ms/step
```

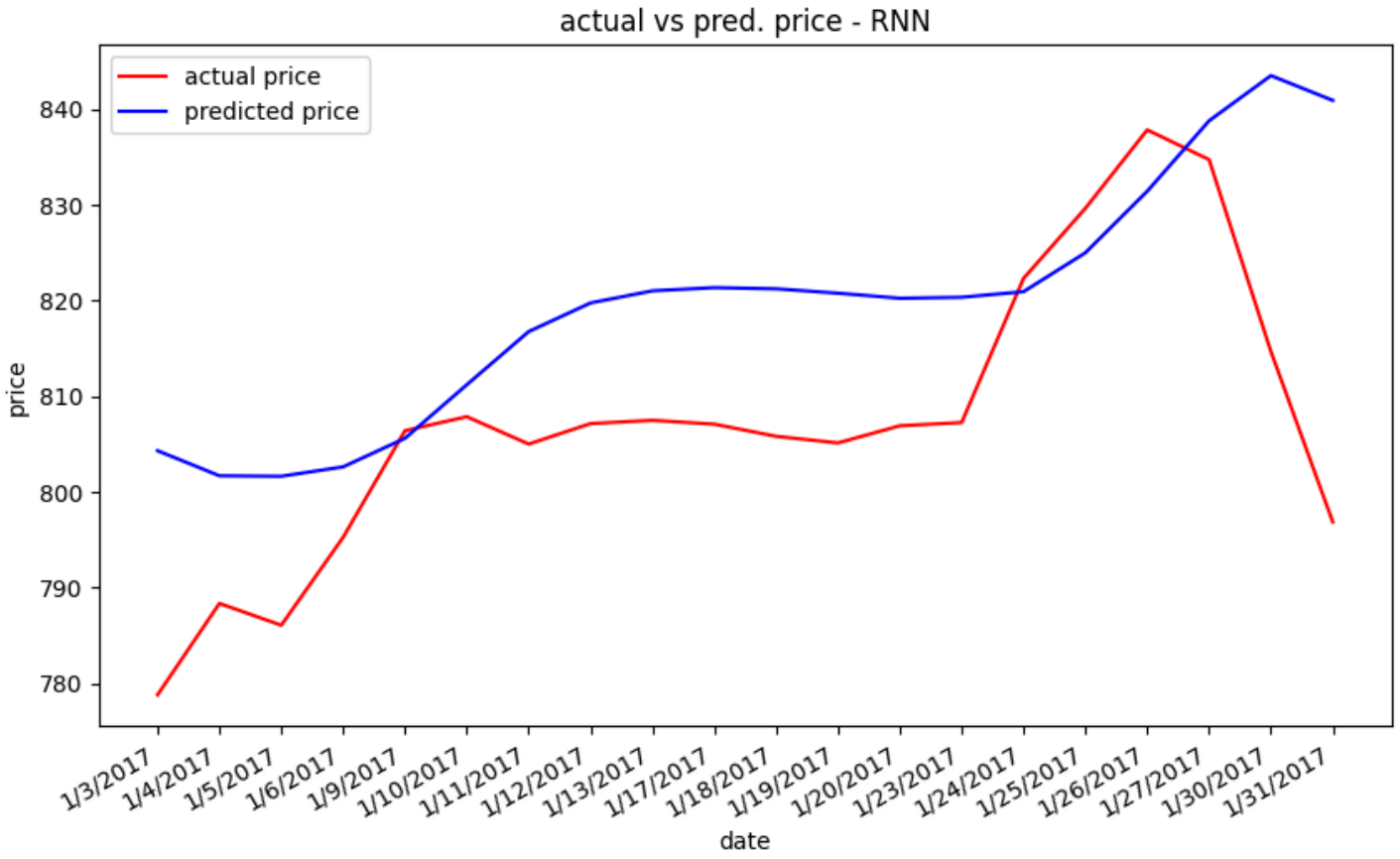
In [ ]:
```
import matplotlib.dates as mdates

# visualize the final results
plt.figure(figsize=(10,6))
plt.plot(
    np.array(df_test.Date),
    y_test_array,
    color='red',
    label='actual price'
)
```

```
plt.plot(
    y_pred_array,
    color='blue',
    label='predicted price'
)

# plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%D'))
# plt.gca().xaxis.set_major_locator(mdates.YearLocator())
plt.gcf().autofmt_xdate()

plt.title('actual vs pred. price - RNN')
plt.xlabel('date')
plt.ylabel('price')
plt.legend()
plt.show()
```



In [ ]:
```
# visualizing the regressor model using keras plot
regressor.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
lstm (LSTM)	(None, 60, 50)	10,400
dropout (Dropout)	(None, 60, 50)	0
lstm_1 (LSTM)	(None, 60, 50)	20,200
dropout_1 (Dropout)	(None, 60, 50)	0
lstm_2 (LSTM)	(None, 60, 50)	20,200
dropout_2 (Dropout)	(None, 60, 50)	0
lstm_3 (LSTM)	(None, 50)	20,200
dropout_3 (Dropout)	(None, 50)	0
dense (Dense)	(None, 1)	51

Total params: 213,155 (832.64 KB)
Trainable params: 71,051 (277.54 KB)
Non-trainable params: 0 (0.00 B)
Optimizer params: 142,104 (555.10 KB)