In [ ]:
```
# Install TensorFlow
# !pip install -q tensorflow-gpu==2.0.0-beta1

try:
  %tensorflow_version 2.x  # Colab only.
except Exception:
  pass

import tensorflow as tf
print(tf.__version__)
```

```
     |████████████████████████████████| 348.9MB 45kB/s
     |████████████████████████████████| 501kB 42.9MB/s
     |████████████████████████████████| 3.1MB 43.5MB/s
2.0.0-beta1
```

In [ ]:
```
from tensorflow.keras.layers import Input, SimpleRNN, Dense, Flatten
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import SGD, Adam

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

In [ ]:
```
# Things you should automatically know and have memorized
# N = number of samples
# T = sequence length
# D = number of input features
# M = number of hidden units
# K = number of output units
```

In [ ]:
```
# Make some data
N = 1
T = 10
D = 3
K = 2
X = np.random.randn(N, T, D)
```

In [ ]:
```
# Make an RNN
M = 5 # number of hidden units
i = Input(shape=(T, D))
x = SimpleRNN(M)(i)
x = Dense(K)(x)

model = Model(i, x)
```

In [ ]:
```
# Get the output
Yhat = model.predict(X)
print(Yhat)
```

```
[[-0.7062384   0.45167243]]
```

In [ ]:
```
# See if we can replicate this output
# Get the weights first
model.summary()
```

```
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 10, 3)]           0
_________________________________________________________________
simple_rnn_1 (SimpleRNN)     (None, 5)                 45
_________________________________________________________________
dense_1 (Dense)              (None, 2)                 12
=================================================================
Total params: 57
Trainable params: 57
Non-trainable params: 0
_________________________________________________________________
```

In [ ]:
```
# See what's returned
model.layers[1].get_weights()
```

Out[ ]:
```
[array([[ 0.06160122,  0.16070706,  0.83621055,  0.04993761, -0.36932853],
        [ 0.4978891 , -0.474034  ,  0.55890614,  0.06967717,  0.21268493],
        [-0.44685632, -0.28297323, -0.17539108,  0.42829865,  0.22275227]],
       dtype=float32),
 array([[ 0.00272548,  0.04928541,  0.32022277,  0.3270029 ,  0.88774437],
        [ 0.6996881 ,  0.64928424, -0.08133215, -0.27187836,  0.09128988],
        [-0.22173485,  0.50949985,  0.6649476 ,  0.31805265, -0.38461757],
        [ 0.5346833 , -0.24025255, -0.13355102,  0.7674074 , -0.22280595],
        [ 0.41877976, -0.50861543,  0.65639263, -0.35927662, -0.07747886]],
       dtype=float32),
 array([0., 0., 0., 0., 0.], dtype=float32)]
```

In [ ]:
```
# Check their shapes
# Should make sense
# First output is input > hidden
# Second output is hidden > hidden
# Third output is bias term (vector of length M)
a, b, c = model.layers[1].get_weights()
print(a.shape, b.shape, c.shape)
```

```
(3, 5) (5, 5) (5,)
```

In [ ]:
```
Wx, Wh, bh = model.layers[1].get_weights()
Wo, bo = model.layers[2].get_weights()
```

In [ ]:
```
h_last = np.zeros(M) # initial hidden state
x = X[0] # the one and only sample
Yhats = [] # where we store the outputs

for t in range(T):
  h = np.tanh(x[t].dot(Wx) + h_last.dot(Wh) + bh)
  y = h.dot(Wo) + bo # we only care about this value on the last iteration
  Yhats.append(y)

  # important: assign h to h_last
```

```
  h_last = h

# print the final output
print(Yhats[-1])
```

```
[-0.70623848  0.45167215]
```

In [ ]:

```
# Bonus exercise: calculate the output for multiple samples at once (N > 1)
```