Recurrent neural nets: intro

Glenn Bruns
CSUMB

Much material in this deck from Géron, Hands-on Machine Learning with Scikit-Learn and TensorFlow

Learning outcomes

After this lecture you should be able to:

- explain concept and applications of recurrent neural networks
- explain and modify a simple RNN in TensorFlow

Recap of neural networks so far

TensorFlow

- computational model and engine
- distribute computation across CPUs/GPUs/cluster

Basic feedforward neural networks

- neuron
- backprop
- optimizer

Convolutional neural networks

- convolutional layers, filters, feature maps
- pooling layers

Recurrent neural networks

Can work on sequences of arbitrary length, rather than fixed-size inputs.

Applications include:

- time-series forecasting
- autonomous vehicles; predict car trajectory
- automatic translation
- speech to text
- generate sentences; generate music

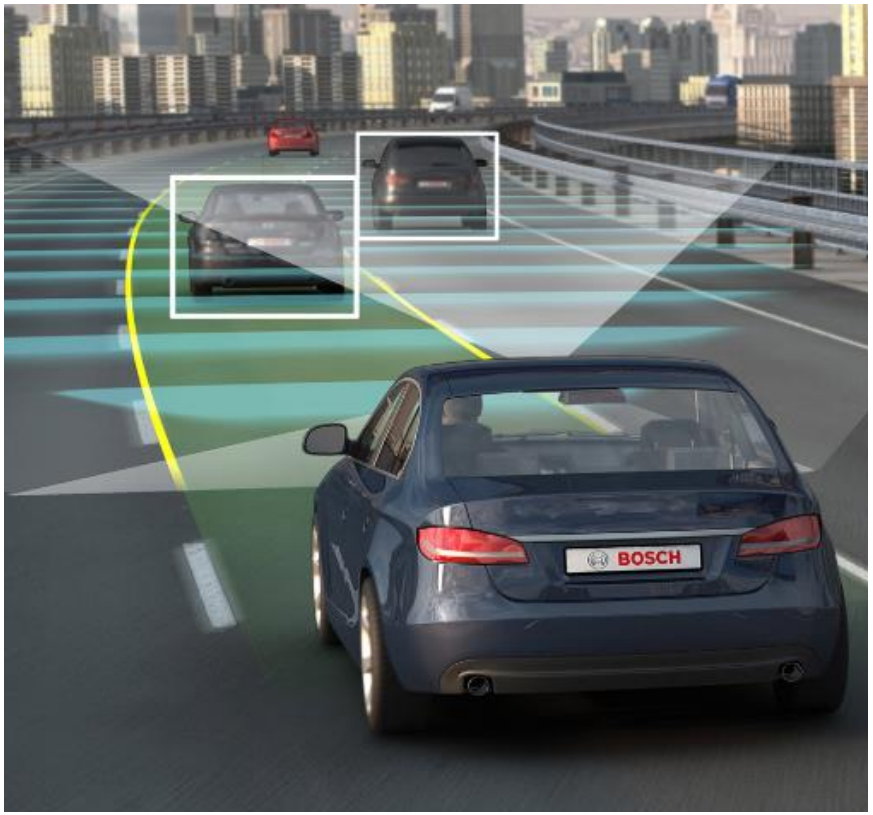


figure: driverlessreport.com/2017/08/autonomous-vehicles-transform-work-travel-live/

Example application

Feed RNN the text of "Alice in Wonderland".

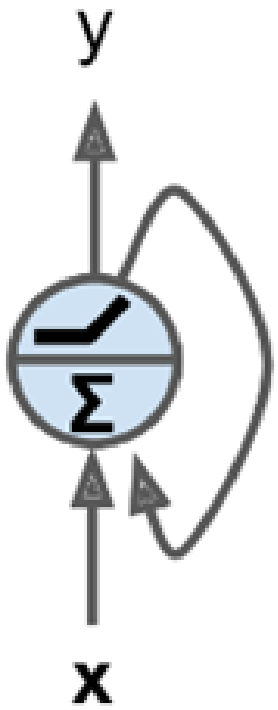
Sample output produced after 40 minutes training:

> As and shout a place gettand in his swe. The Queen.
>
> 'They were speak.'
>
> 'Os, thes hack a minute or do look of mesteren your D

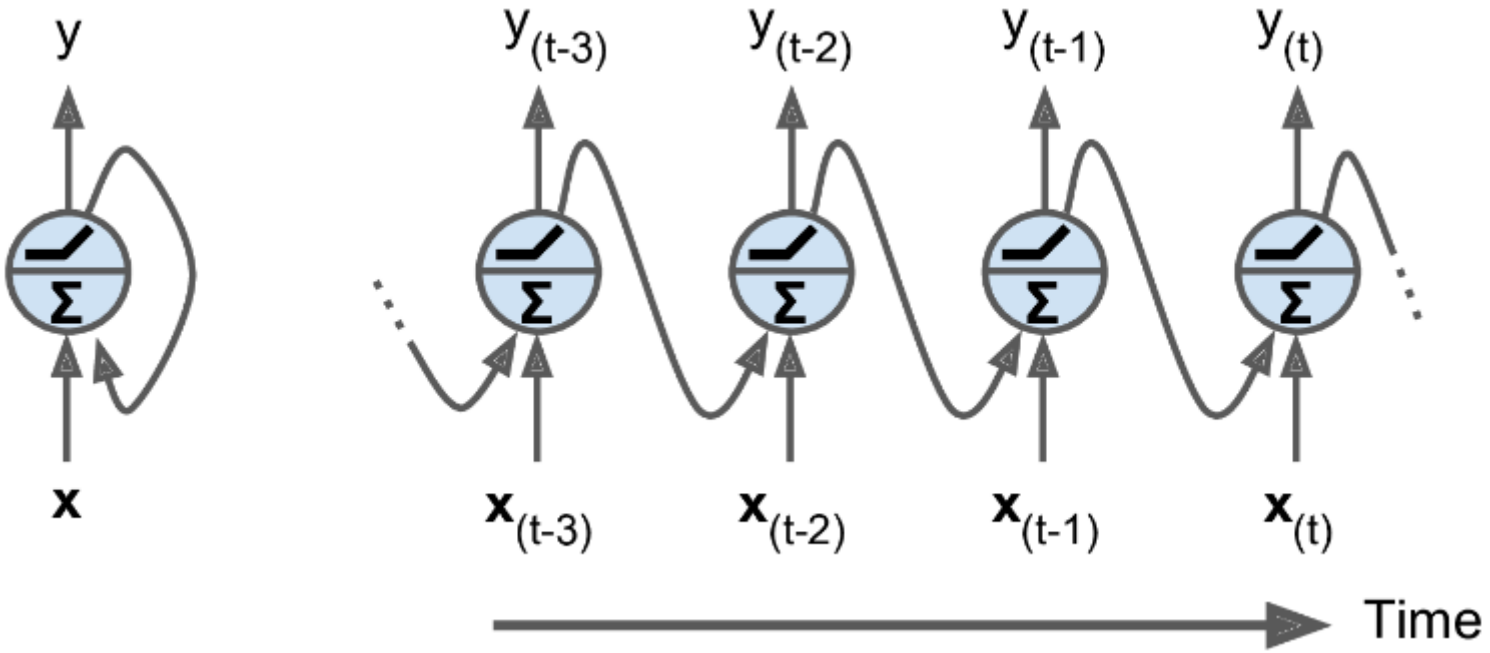
A recurrent neuron

Recurrent neural networks (RNNs) have connections that "point" backwards

At each time step, the recurrent neuron receives input x, as well as output from the previous step.



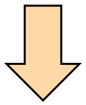
Unrolling a RNN



Unrolling a recurrent neuron shows the time relationships.

Analogy: unrolling code

```
sum = 0
for i in range(50):
   sum = sum + i
```



unrolled:

```
sum = 0
sum = sum + 1
sum = sum + 2
…
sum = sum + 50
```

```
h = 0
for i in range(50):
   h = f(x[i], h)
   y[i] = g(h)
```

unrolled:

```
h = 0

h = f(x[0], h)
y[0] = g(h)

h = f(x[1], h)
y[1] = g(h)
…
h = f(x[49], h)
y[49] = g(h)
```

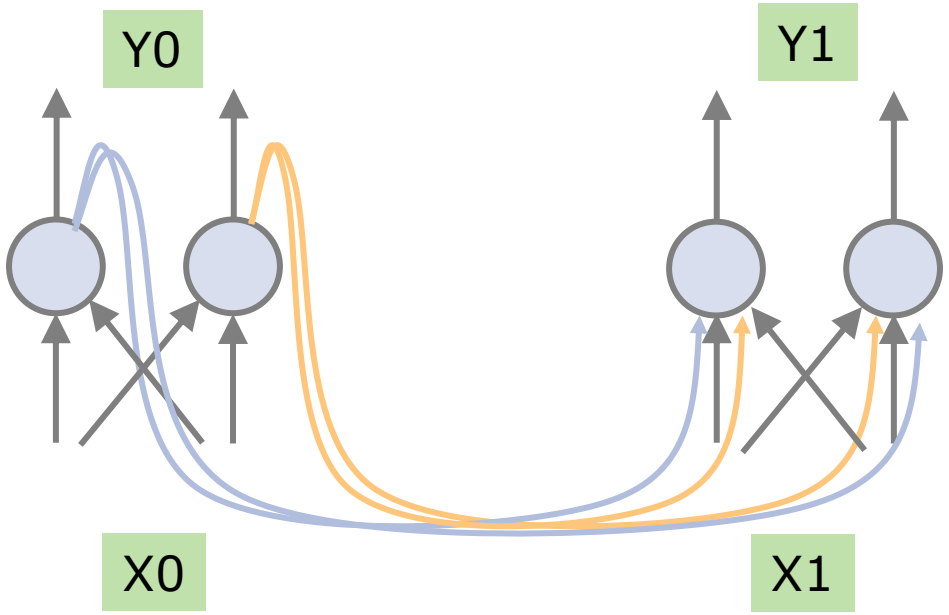
alternative unrolled:

```
h[-1] = 0

h[0] = f(x[0], h[-1])
y[0] = g(h[0])

h[1] = f(x[1], h[0])
y[1] = g(h[1])
…
h[49] = f(x[49], h[48])
y[49] = g(h[49])
```

Unrolling multiple neurons



Unrolling one time step for two recurrent neurons

Output of a recurrent layer

$$\mathbf{y}_{(t)} = \phi\left(\mathbf{W}_x^T \cdot \mathbf{x}_{(t)} + \mathbf{W}_y^T \cdot \mathbf{y}_{(t-1)} + \mathbf{b}\right)$$

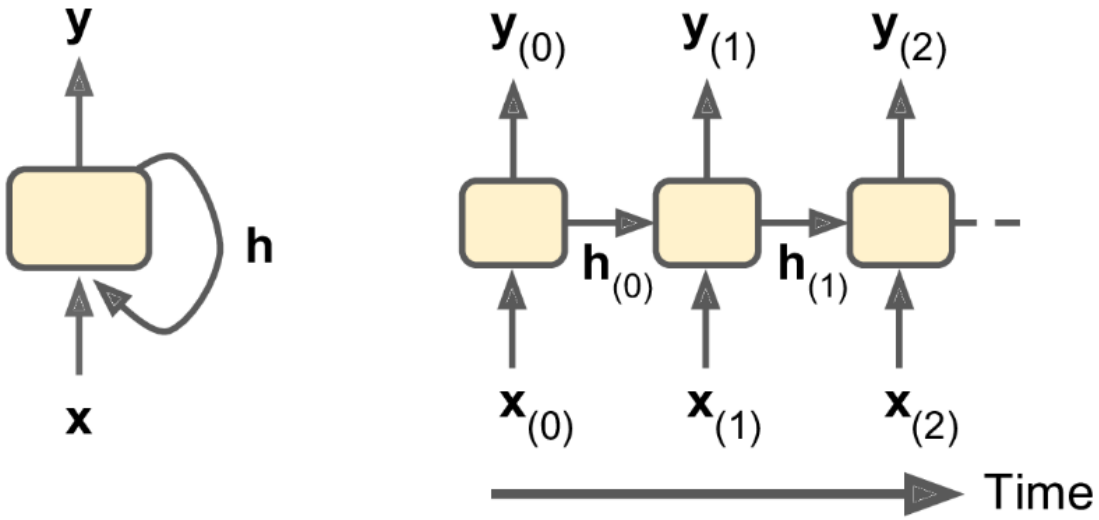
- two sets of weights: one for inputs, and one for feedback (output of previous step)
- b is bias term

Memory cells

The output of a recurrent neuron is a function of data from all past time steps

In this sense the recurrent neuron has "memory"

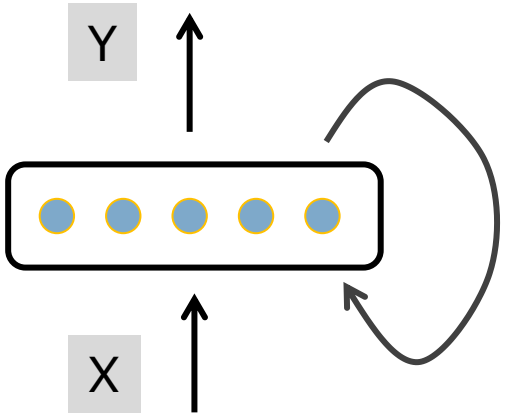
A part of a net that preserves state across time steps is called a memory cell ('cell' for short)



Simple RNN example

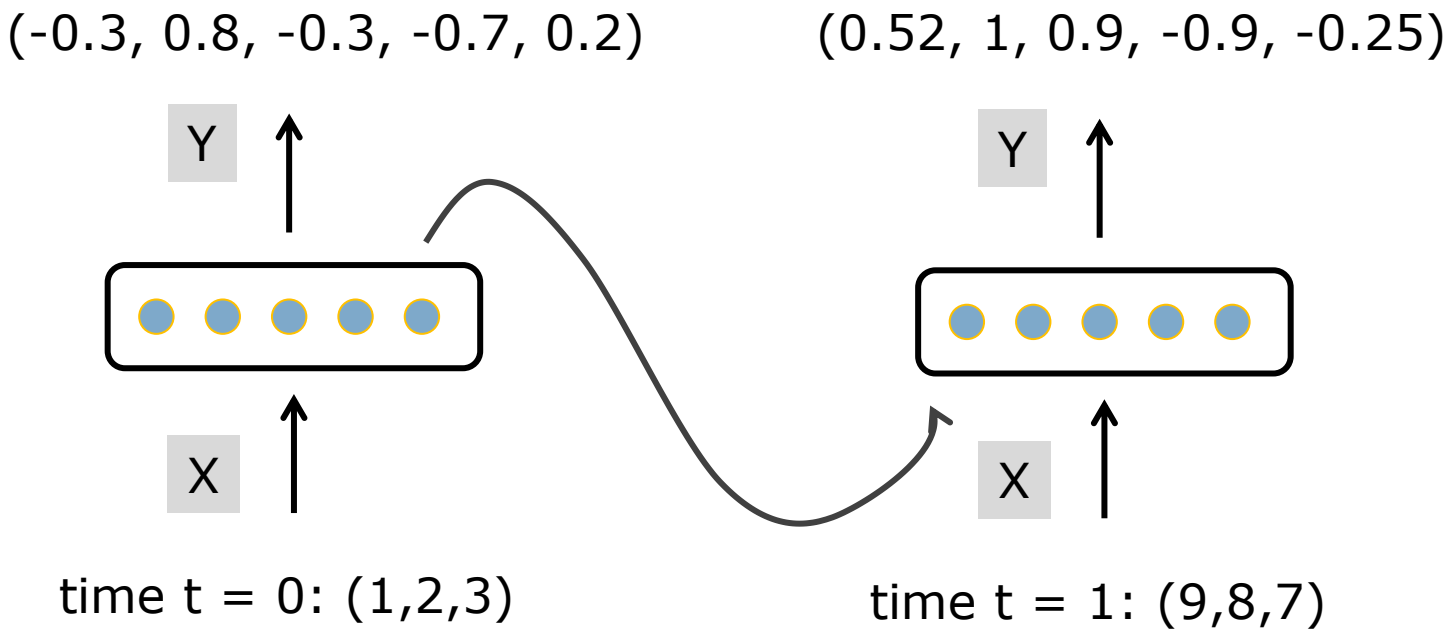
An RNN cell of 5 neurons

time t = 0: (-0.3, 0.8, -0.3, -0.7, 0.2)
t = 1: (0.52, 1, 0.9, -0.9, -0.25)



time t = 0: (1,2,3)
t = 1: (9,8,7)

Example unrolled



A manually-created RNN

```
n_inputs = 3          # inputs per time step
n_neurons = 5         # neurons per recurrent cell

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

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))

# the RNN is manually unrolled over two time steps
Y0 = tf.tanh(tf.matmul(X0, Wx) + b)  # no previous output
Y1 = tf.tanh(tf.matmul(Y0, Wy) + tf.matmul(X1, Wx) + b)

init = tf.global_variables_initializer()
```

Execution of the RNN

```
# mini-batch:     instance 0, instance 1, instance 2, instance 3
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

with tf.Session() as sess:
    init.run()
    Y0_val, Y1_val = sess.run([Y0, Y1],
                              feed_dict={X0: X0_batch, X1: X1_batch})
```

output:

```
>>> print(Y0_val)   # output at t = 0

[[-0.0664006   0.96257669  0.68105793  0.70918542 -0.89821601]
 [ 0.9977755  -0.71978903 -0.99657607  0.96739239 -0.99989718]
 [ 0.99999774 -0.99898803 -0.99999893  0.99677622 -0.99999988]
 [ 1.         -1.         -1.         -0.99818915  0.99950868]]
```

Summary

- in a recurrent neuron, output from past time step as used as input
- "unroll" a recurrent neuron to see inputs and outputs over time
- you can build a simple RNN manually in TensorFlow by unrolling