Recurrent neural nets: Time series predictions

Glenn Bruns
CSUMB

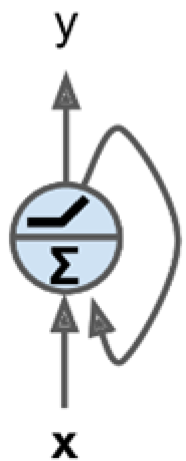
Much material in this deck from Géron, Hands-on Machine Learning with Scikit-Learn and TensorFlow

Learning outcomes

After this lecture you should be able to:

- build an RNN that performs time series predictions

RNN basics review



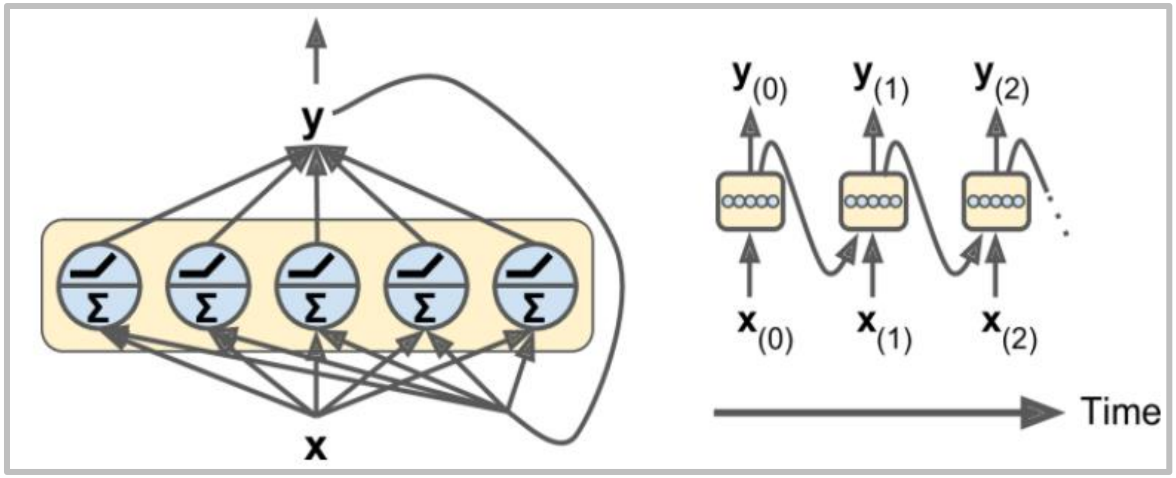
Suppose a training instance is 20 inputs long

Each input has only one feature

How many weights associated with the recurrent neuron?

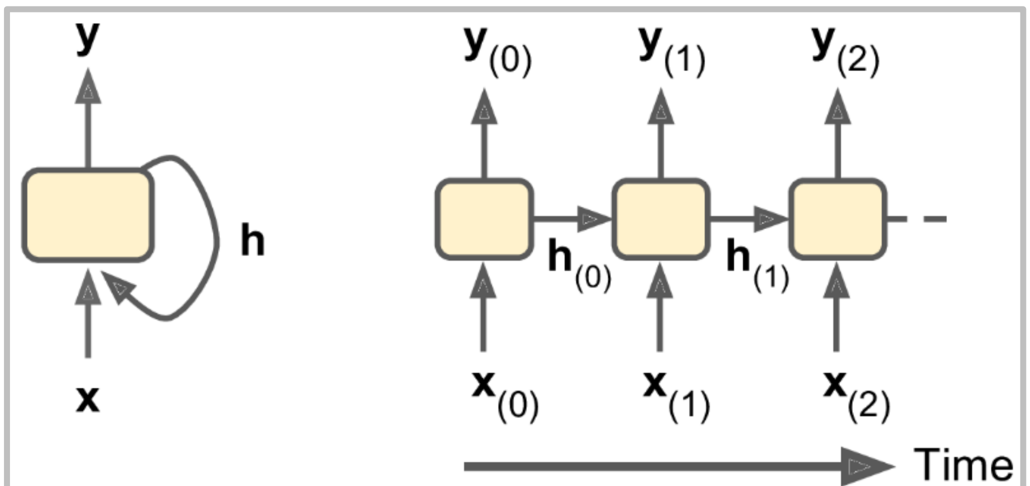
time step t	x(t)	y(t)
0	2	$\phi(2 \cdot w_x + 0 \cdot w_y + b)$
1	3	$\phi(3 \cdot w_x + y(0) \cdot w_y + b)$
2	2	…
3	5	…
4	6	…

Recurrent layers and memory cells



a layer of recurrent neurons

number of neurons in layer has nothing to do with length of input sequence



A cell is one recurrent neuron, a layer of recurrent neurons, or a larger part of a neural net

$\mathbf{h}_{(t)}$ represents a cell's state at time step t.

Input and output sequences

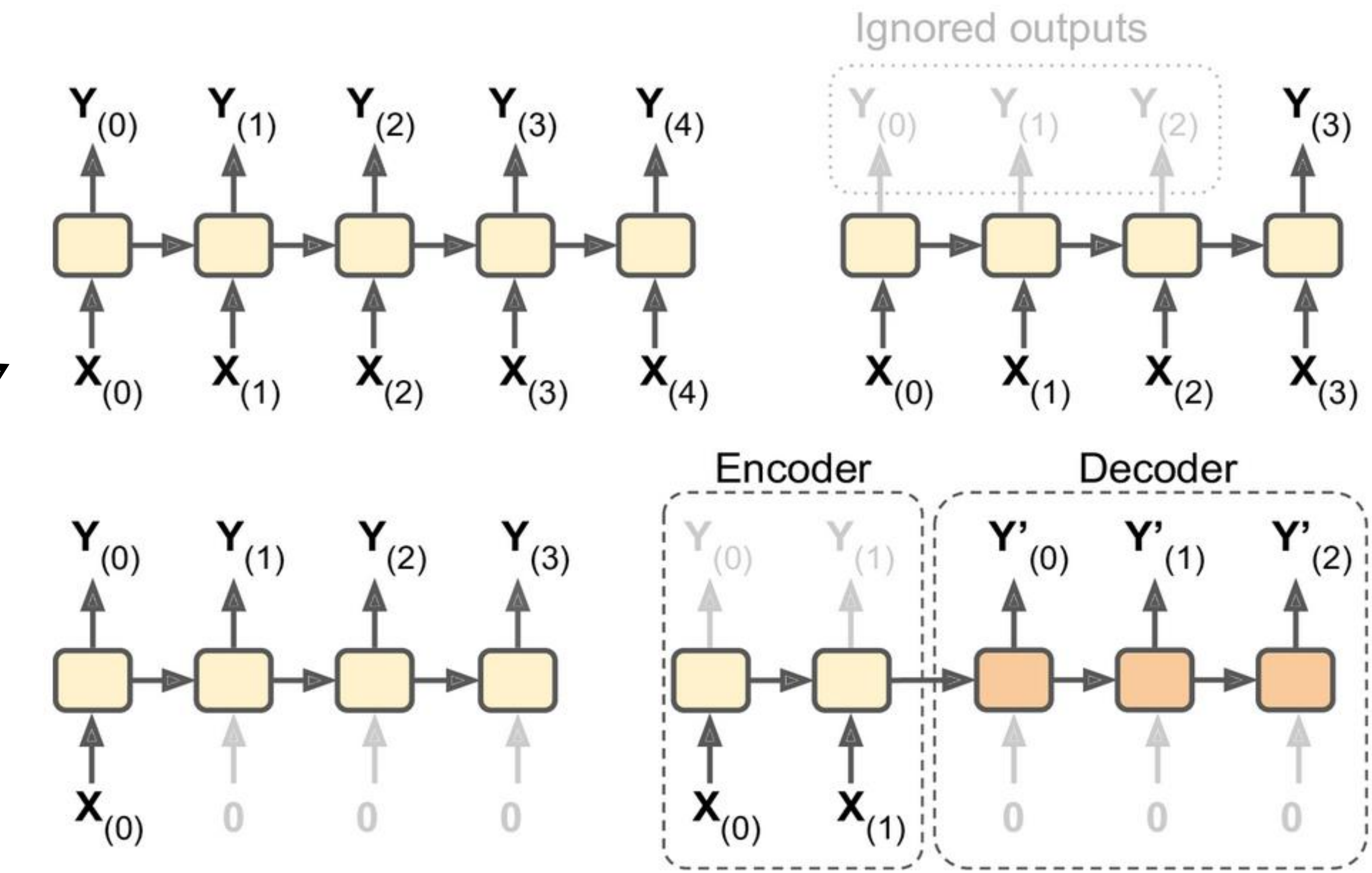
An RNN can map a sequence of inputs to a sequence of inputs, or other variants.

seq-to-seq

seq-to-vec

vec-to-sec

sec-to-vec-to-sec

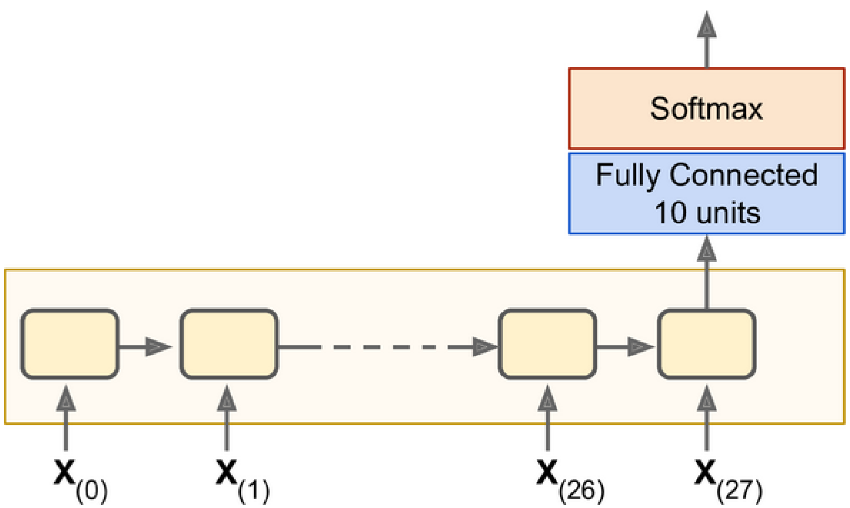


RNN MNIST classifier (review)

We treat an 28 x 28 pixel image as a sequence of 28 rows, each with 28 pixels

The TF RNN has:

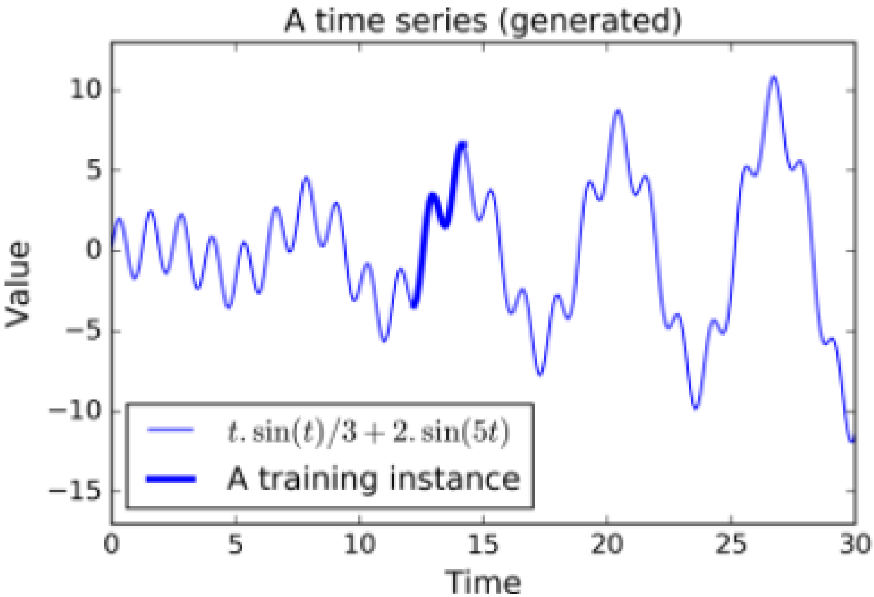
- cells of 150 recurrent neurons
- a fully connected layer of 10 neurons

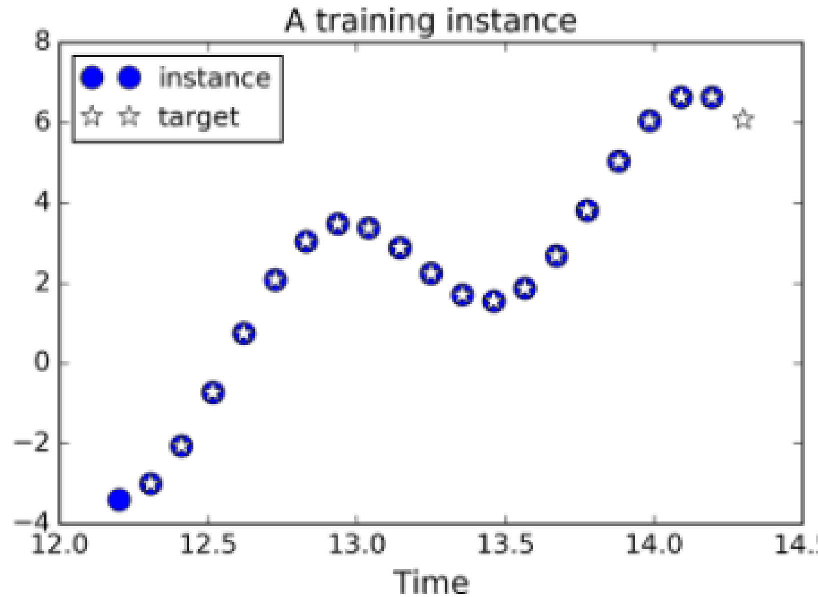


What is the shape of $X_{(1)}$?

How does the recurrent layer connect to the fully connected layers?

Time series forecasting with RNNs





- training instance: 20 values from the series
- each input has only one feature
- target: 20 values, but shifted over by one step
- 100 recurrent neurons

Construct the RNN

```
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])
cell = tf.contrib.rnn.BasicRNNCell(num_units=n_neurons,
                                   activation=tf.nn.relu)
outputs, states = tf.nn.dynamic_rnn(cell, X, dtype=tf.float32)
```

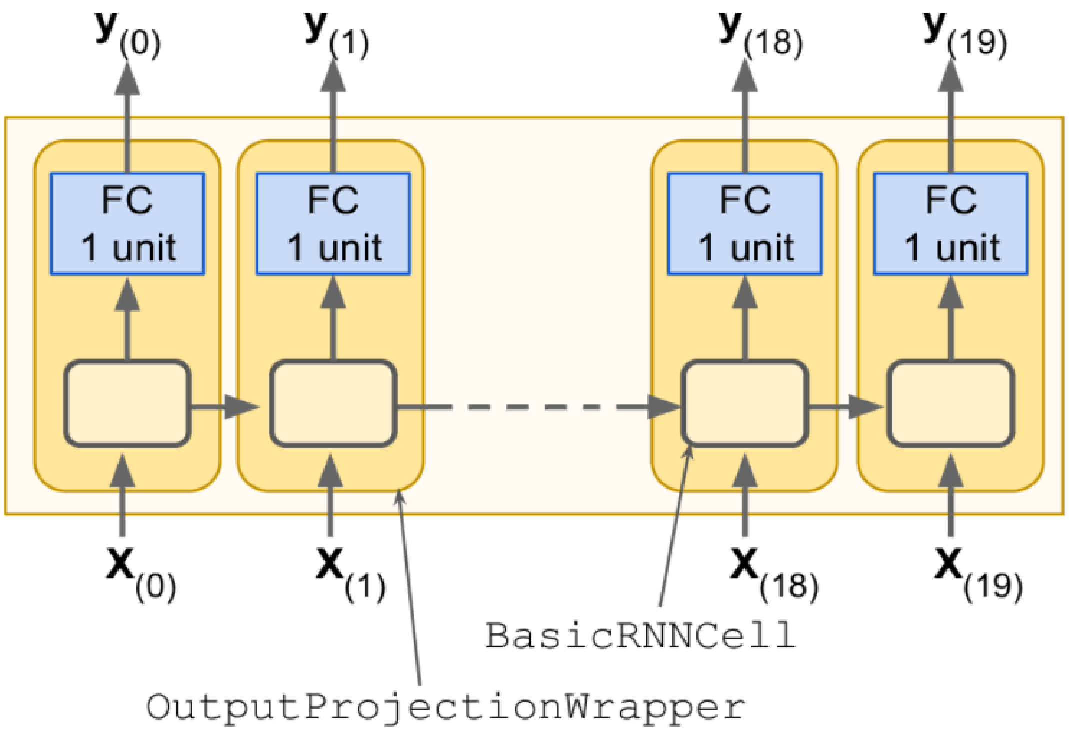
What is the size of the output vector at each time step?

Why does it say n_outputs = 1?

```
In [9]: outputs.get_shape()
Out[9]: TensorShape([Dimension(None), Dimension(20), Dimension(100)])
```

Output projections

```
cell = tf.contrib.rnn.OutputProjectionWrapper(
    tf.contrib.rnn.BasicRNNCell(num_units=n_neurons, activation=tf.nn.relu),
    output_size=n_outputs)
```



The wrapper adds a fully-connected (FC) layer of linear neurons on top of each output.

('linear' means no activation function.)

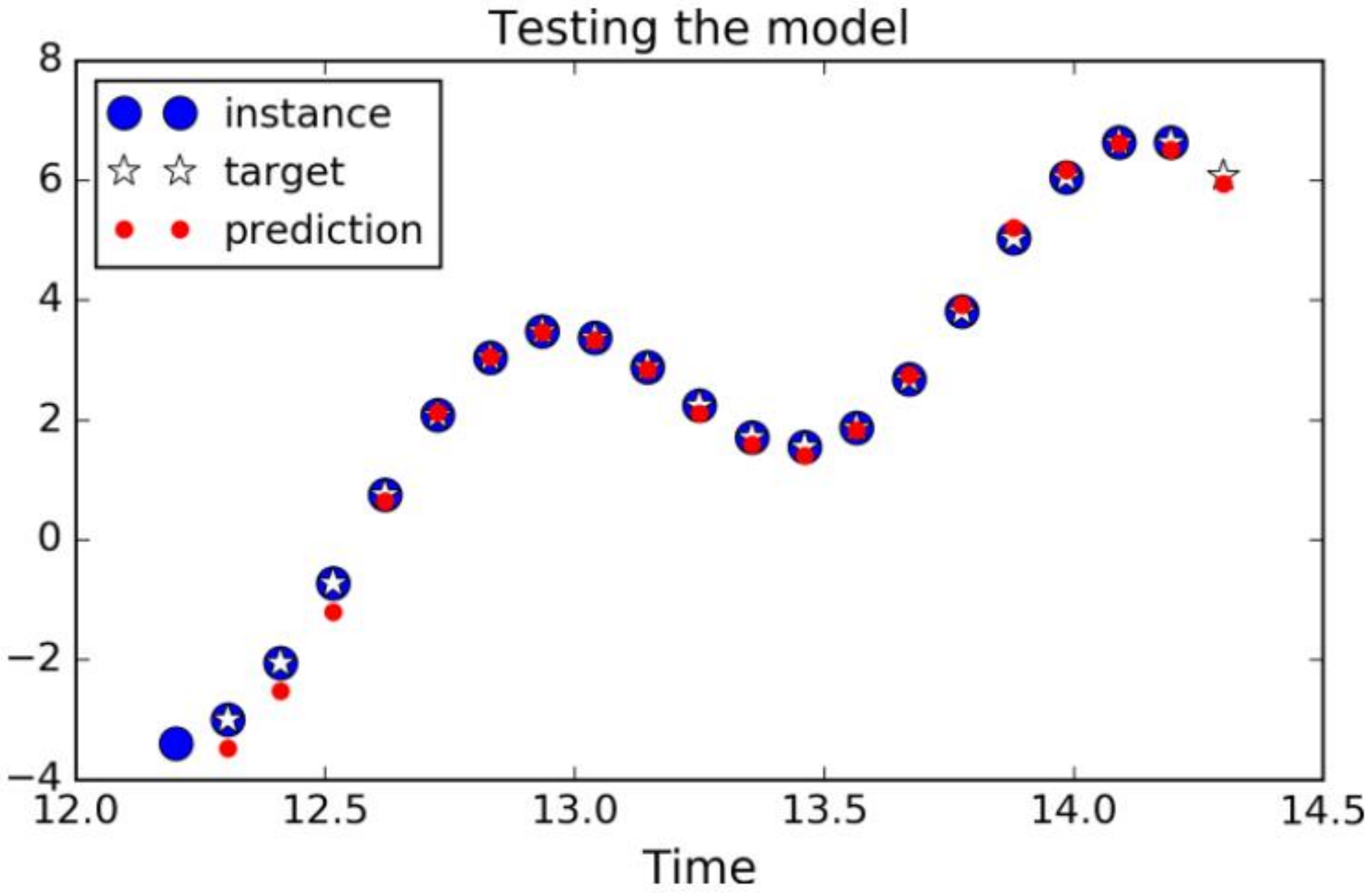
```
In [11]: outputs.get_shape()
Out[11]: TensorShape([Dimension(None), Dimension(20), Dimension(1)])
```

Cost function and execution phase

```
learning_rate = 0.001

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()
```

```
n_iterations = 1500
batch_size = 50

with tf.Session() as sess:
  init.run()
  for iteration in range(n_iterations):
    X_batch, y_batch = [...]  # fetch the next training batch
    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)
```

Predictions



Summary

- review of RNN structure
- doing time series prediction with RNNs
  - OutputProjectionWrapper