Recurrent Neural Networks

Prof. Gheith Abandah

Reference: *Hands-On Machine Learning with Scikit-Learn and TensorFlow* by Aurélien Géron (O'Reilly), 2017, 978-1-491-96229-9.

Introduction

- YouTube Video: *Deep Learning with Tensorflow - The Recurrent Neural Network Model* from Cognitive Class

https://youtu.be/C0xoB8L8ms0

Outline

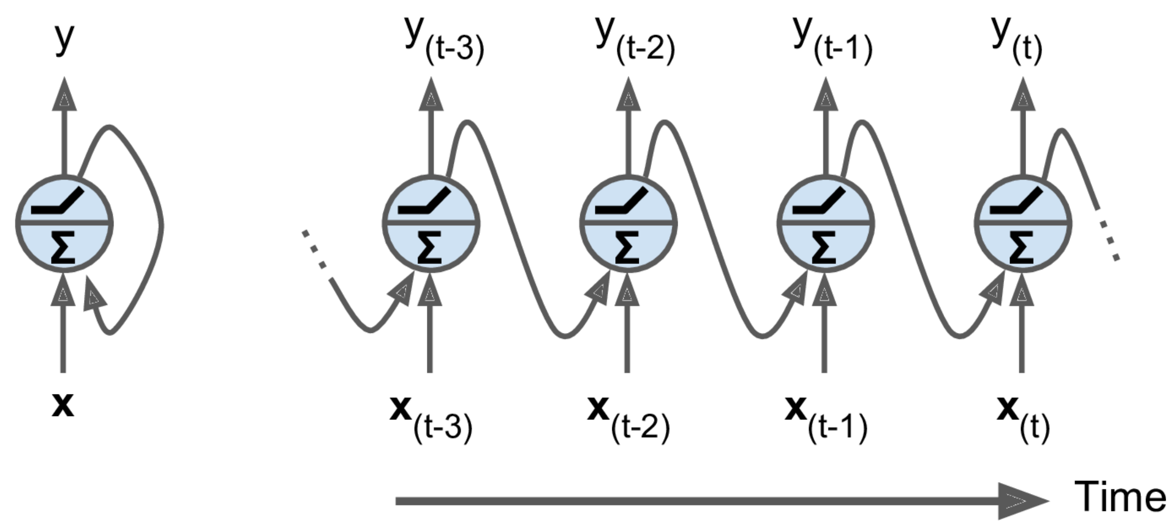
1. Introduction
2. Recurrent neurons
3. Input and output sequences lengths
4. Training RNNs
    1. Predicting time series example
    2. Sequence classifier example of MNIST images
5. Deep RNNs
6. LSTM and GRU cells
7. Exercises

1. Introduction

- *Recurrent neural networks (RNNs)* are used to handle time series data or sequences.
- Applications:
  - Predicting the future (stock prices)
  - Autonomous driving systems (predicting trajectories)
  - Natural language processing (automatic translation, speech-to-text, or sentiment analysis)
  - Creativity (music composition, handwriting, drawing)
  - Image analysis (image captions)

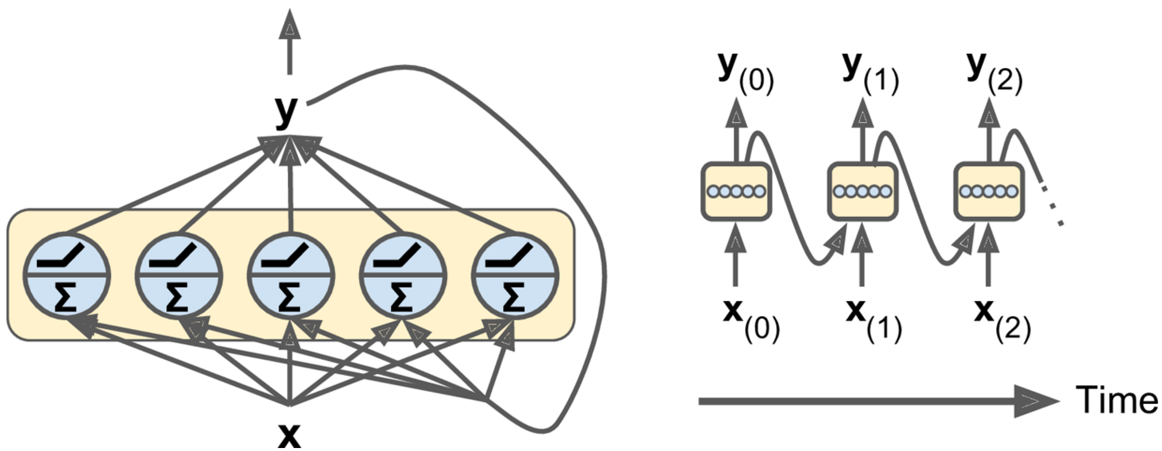
2. Recurrent Neurons

- The figure below shows a *recurrent neuron* (left), unrolled through time (right).



2. Recurrent Neurons

- Multiple recurrent neurons can be used in a layer.

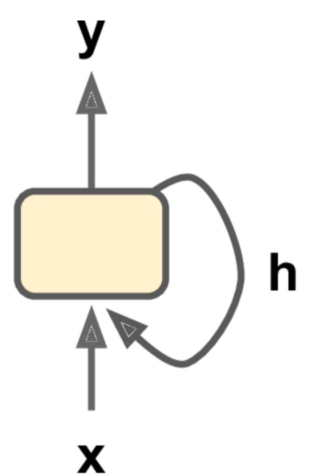


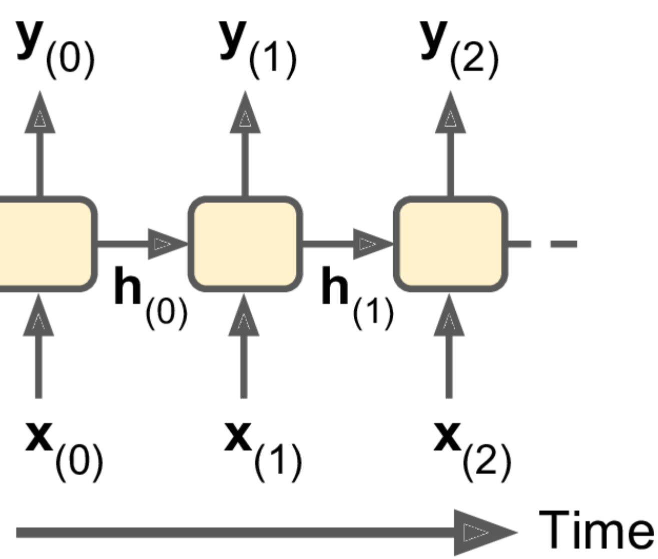
- The output of the layer is:

$$\mathbf{Y}_{(t)} = \phi\left(\mathbf{X}_{(t)} \cdot \mathbf{W}_x + \mathbf{Y}_{(t-1)} \cdot \mathbf{W}_y + \mathbf{b}\right)$$

2. Recurrent Neurons

- Recurrent neurons have memory (hold state) and are called *memory cells*.
- The state $\mathbf{h}_{(t)} = f(\mathbf{h}_{(t-1)}, \mathbf{x}_{(t)})$, not always $\equiv \mathbf{y}_{(t)}$

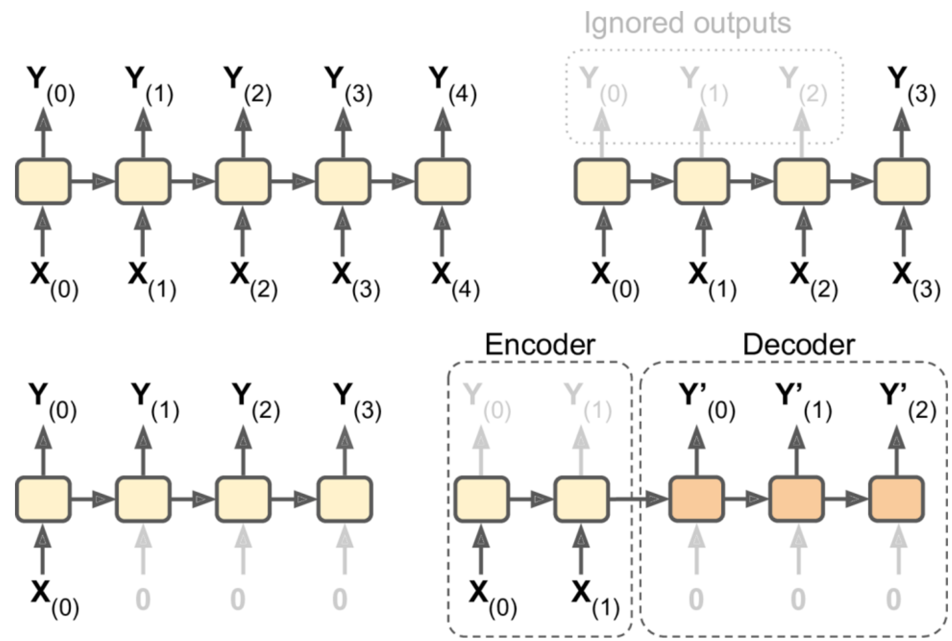




3. Input and Output Sequences Lengths

1. **Seq to seq**: For predicting the future.
2. **Seq to vector**: For analysis, e.g., sentiment score.
3. **Vector to seq**: For image captioning.
4. **Delayed seq to seq**: For sequence transcription.

1. 2.



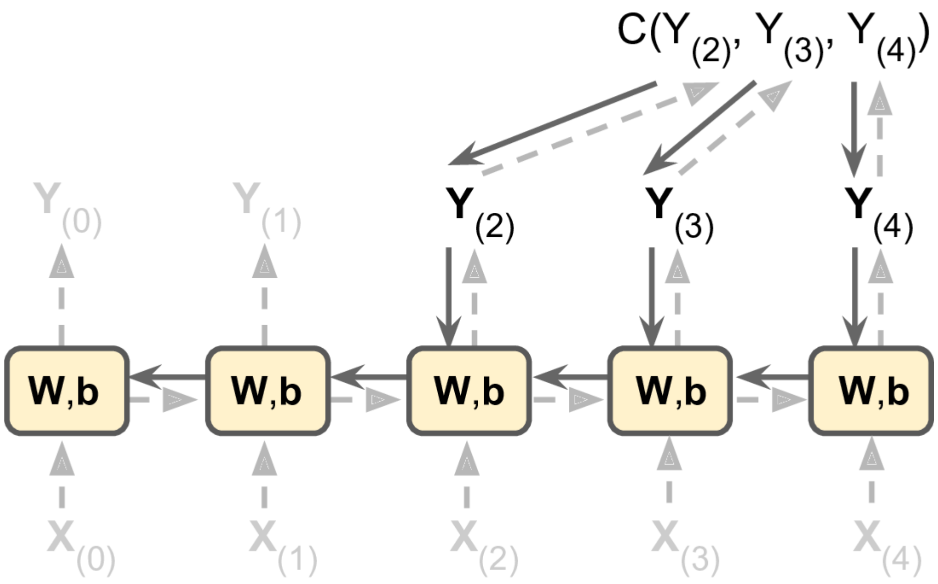
3. 4.

Outline

1. Introduction
2. Recurrent neurons
3. Input and output sequences lengths
4. Training RNNs
    1. Predicting time series example
    2. Sequence classifier example of MNIST images
5. Deep RNNs
6. LSTM and GRU cells
7. Exercises

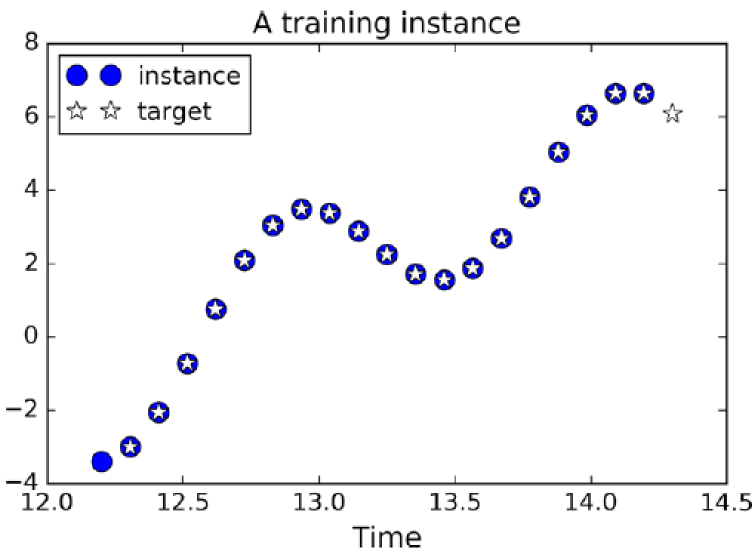
4. Training RNNs

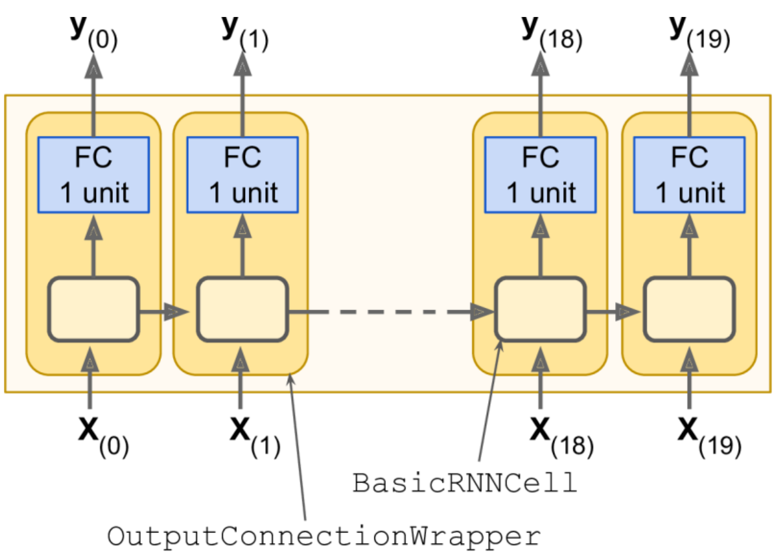
- Training using strategy called *backpropagation through time* (BPTT).
- Forward pass (dashed)
- Cost function of the not-ignored outputs.
- Cost gradients are propagated backward through the unrolled network.



4. Training RNNs: Example 1

- Example: Predict time series.
- Use 100 RNN cells and one fully-connected output neuron.





4. Training RNNs: Example 1

Wraps 100 RNN cells to one output.

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

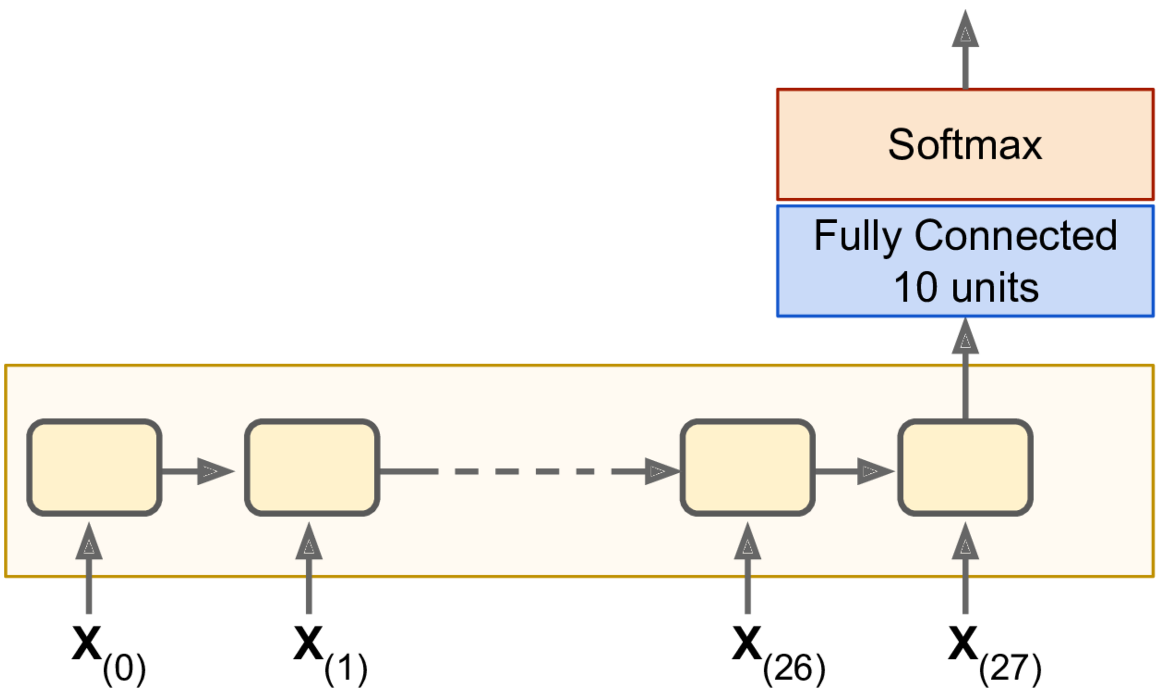
4. Training RNNs: Example 1

```
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 = 10000
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)
```

```
0     MSE: 379.586
100   MSE: 14.58426
200   MSE: 7.14066
300   MSE: 3.98528
400   MSE: 2.00254
[...]
```

4. Training RNNs: Example 2

- Example: training a sequence classifier of MNIST images.



4. Training RNNs: Example 2

```
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.contrib.rnn.BasicRNNCell(num_units=n_neurons)
outputs, states = tf.nn.dynamic_rnn(basic_cell, X, dtype=tf.float32)

logits = fully_connected(states, n_outputs, activation_fn=None)
```

Performs fully dynamic unrolling of inputs.
Returns all outputs and final states.

4. Training RNNs: Example 2

```
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()

from tensorflow.examples.tutorials.mnist import input_data

mnist = input_data.read_data_sets("/tmp/data/")
X_test = mnist.test.images.reshape((-1, n_steps, n_inputs))
y_test = mnist.test.labels
```

4. Training RNNs: Example 2

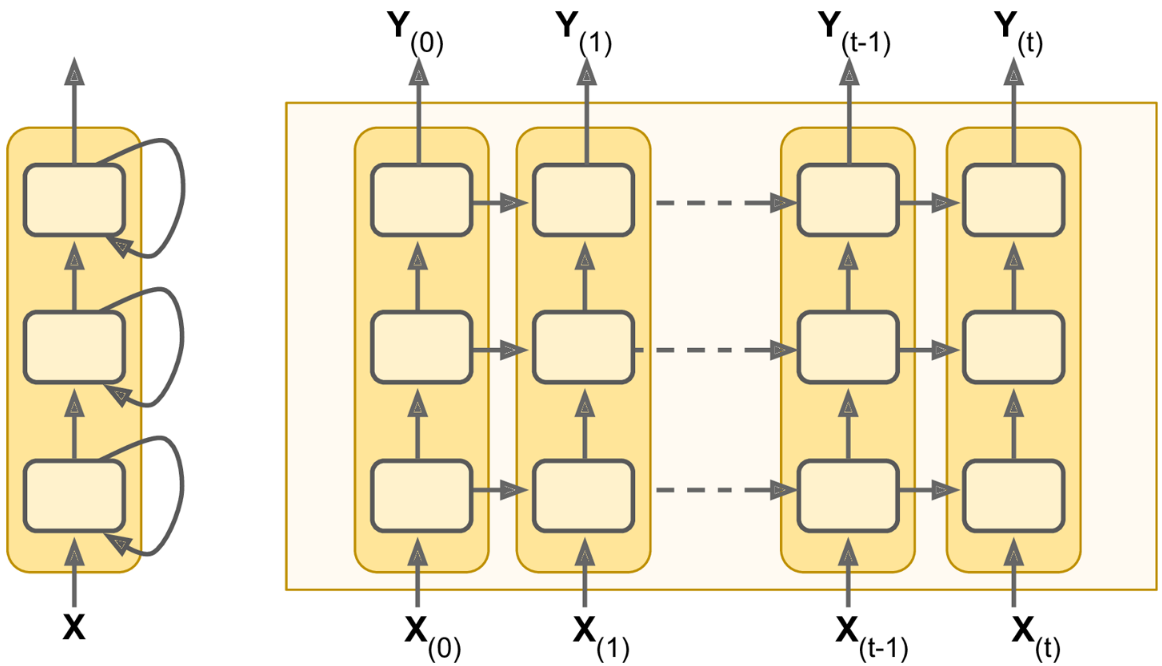
```
n_epochs = 100
batch_size = 150

with tf.Session() as sess:
    init.run()
    for epoch in range(n_epochs):
        for iteration in range(mnist.train.num_examples // batch_size):
            X_batch, y_batch = mnist.train.next_batch(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_train = accuracy.eval(feed_dict={X: X_batch, y: y_batch})
        acc_test = accuracy.eval(feed_dict={X: X_test, y: y_test})
        print(epoch, "Train accuracy:", acc_train, "Test accuracy:", acc_test)
```

```
0 Train accuracy: 0.713333 Test accuracy: 0.7299
1 Train accuracy: 0.766667 Test accuracy: 0.7977
...
98 Train accuracy: 0.986667 Test accuracy: 0.9777
99 Train accuracy: 0.986667 Test accuracy: 0.9809
```

Outline

1. Introduction
2. Recurrent neurons
3. Input and output sequences lengths
4. Training RNNs
    1. Predicting time series example
    2. Sequence classifier example of MNIST images
5. Deep RNNs
6. LSTM and GRU cells
7. Exercises

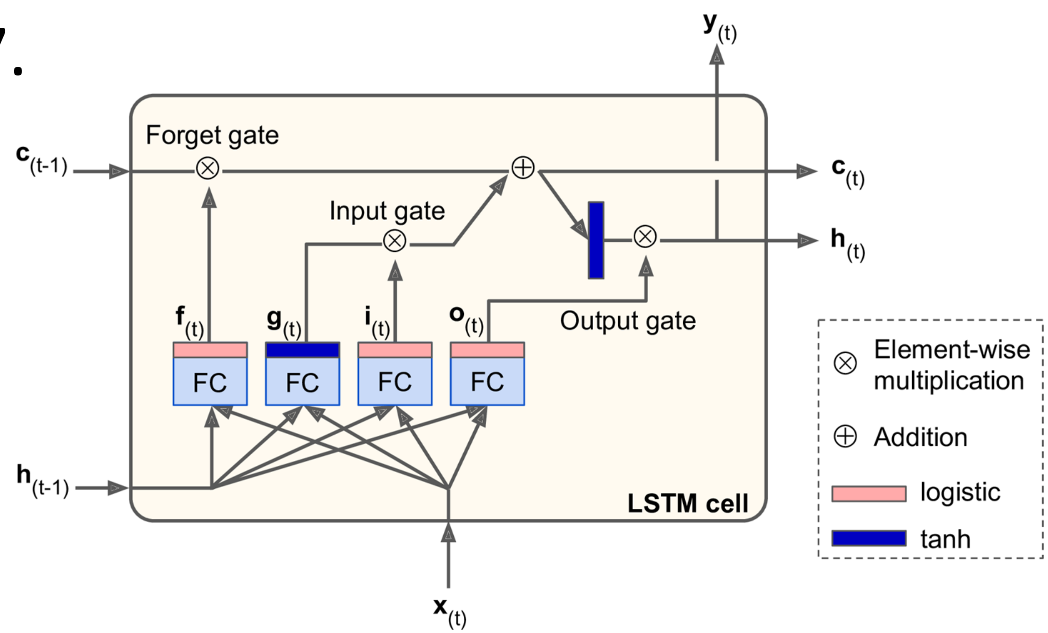
5. Deep RNNs




```
n_neurons = 100
n_layers = 3

basic_cell = tf.contrib.rnn.BasicRNNCell(num_units=n_neurons)
multi_layer_cell = tf.contrib.rnn.MultiRNNCell([basic_cell] * n_layers)
outputs, states = tf.nn.dynamic_rnn(multi_layer_cell, X, dtype=tf.float32)
```

6. LSTM Cell

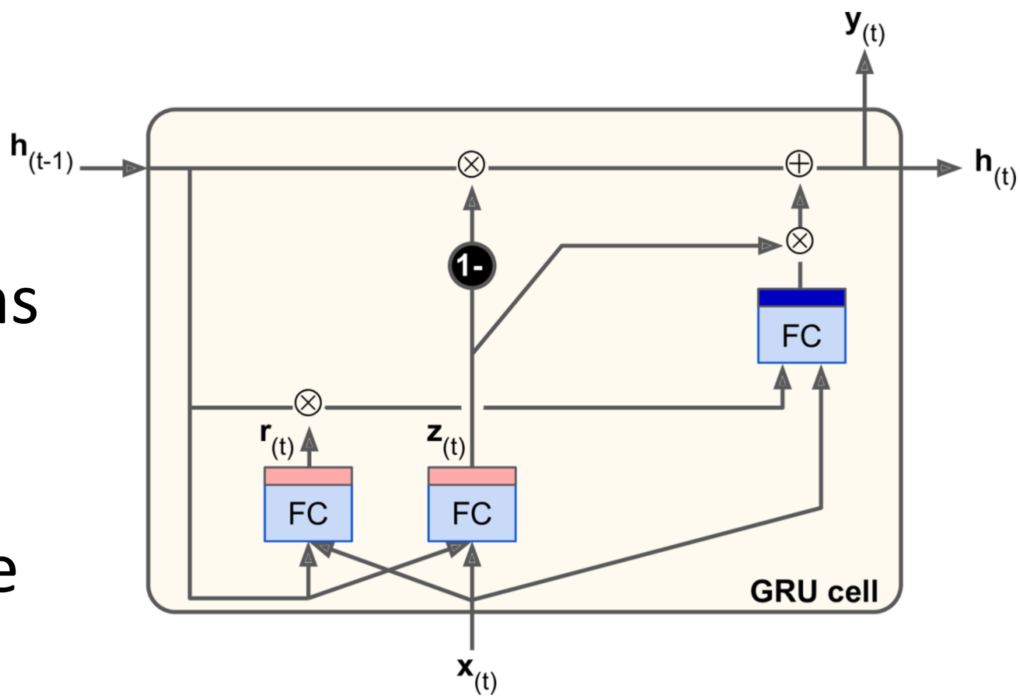
- The *Long Short-Term Memory* (LSTM) cell was proposed in 1997.
- Training converges faster and it detects long-term dependencies in the data.
- $\mathbf{h}_{(t)}$ as the short-term state and $\mathbf{c}_{(t)}$ as the long-term state.




```
lstm_cell = tf.contrib.rnn.BasicLSTMCell(num_units=n_neurons)
```

6. GRU Cell

- The *Gated Recurrent Unit* (GRU) cell was proposed in 2014.
- Simplified version of the LSTM cell, performs just as well.
- A single gate controls the forget gate and the input gate.




```
gru_cell = tf.contrib.rnn.GRUCell(num_units=n_neurons)
```

Summary

1. Introduction
2. Recurrent neurons
3. Input and output sequences lengths
4. Training RNNs
    1. Predicting time series example
    2. Sequence classifier example of MNIST images
5. Deep RNNs
6. LSTM and GRU cells
7. Exercises

Exercises

From Chapter 14, solve exercises:

- 2
- 3
- 8