Using RNN in a forecasting problem

Santiago Morales[1]

[1]physics institute
Universidad de Antioquia

Medellin, 2018

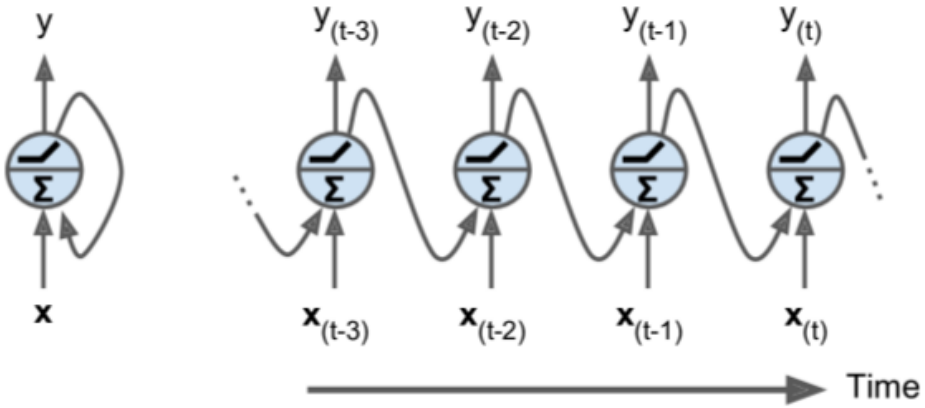
Recurrent networks

1. Theory
   - Recurrent cell
   - Long short-term memory cell
2. Context
   - Problem statement
   - Data information
3. Implementation using TensorFlow
   - Code
   - Alternative Results
   - Results using RNN

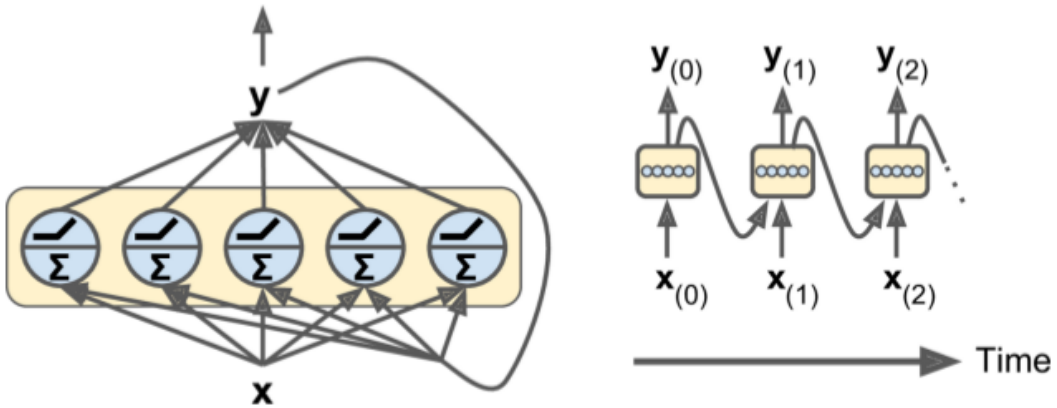
Recurrent networks

Recurrent cell

- Recurrent cell unwrapped in time:



- Recurrent layer unwrapped in time:

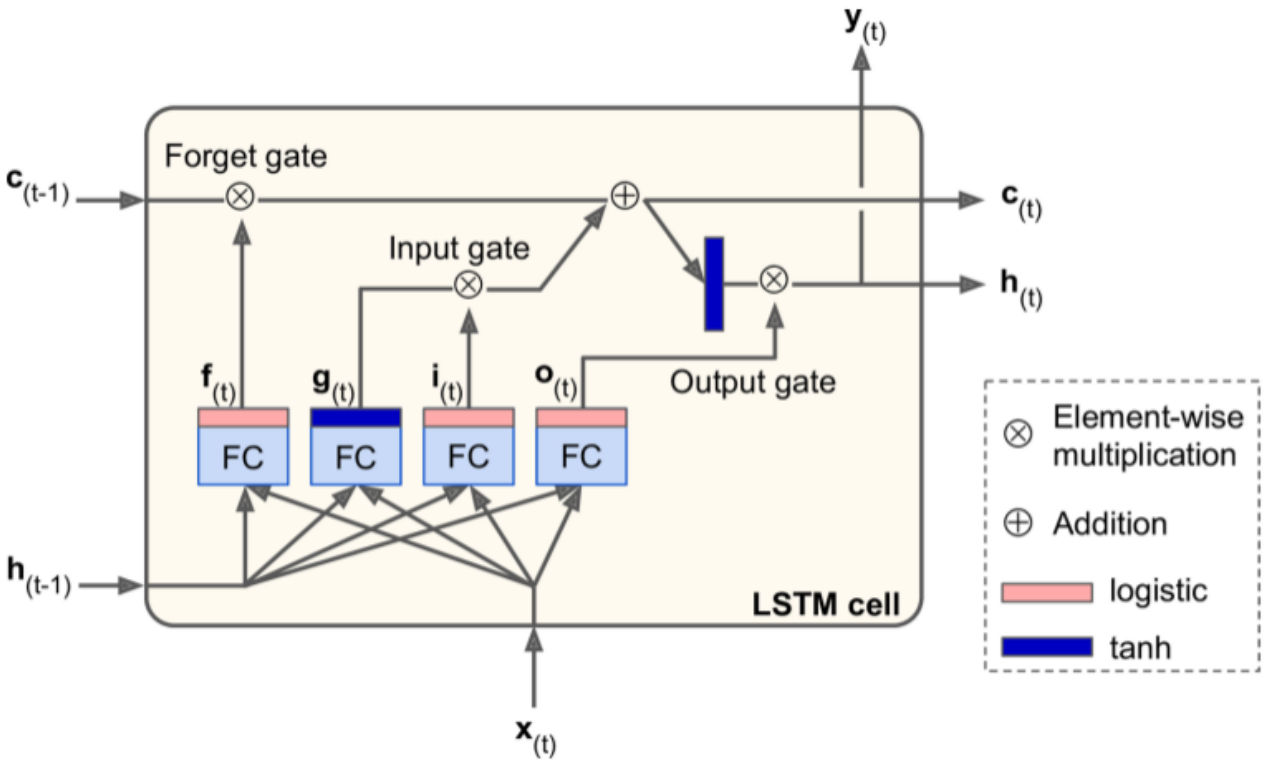


Recurrent networks

1. Theory
   - Recurrent cell
   - Long short-term memory cell
2. Context
   - Problem statement
   - Data information
3. Implementation using TensorFlow
   - Code
   - Alternative Results
   - Results using RNN

LSTM cell

- *LSTM* cell



Recurrent networks

Problem statement

- Problem Statement:
  We know the sales of 21807 products in 60 stores between January 2013 and October 2015, and we want to predict the quantity of each product that will be sold in each store in November 2015. This problem was taken from Kaggle.

Recurrent networks

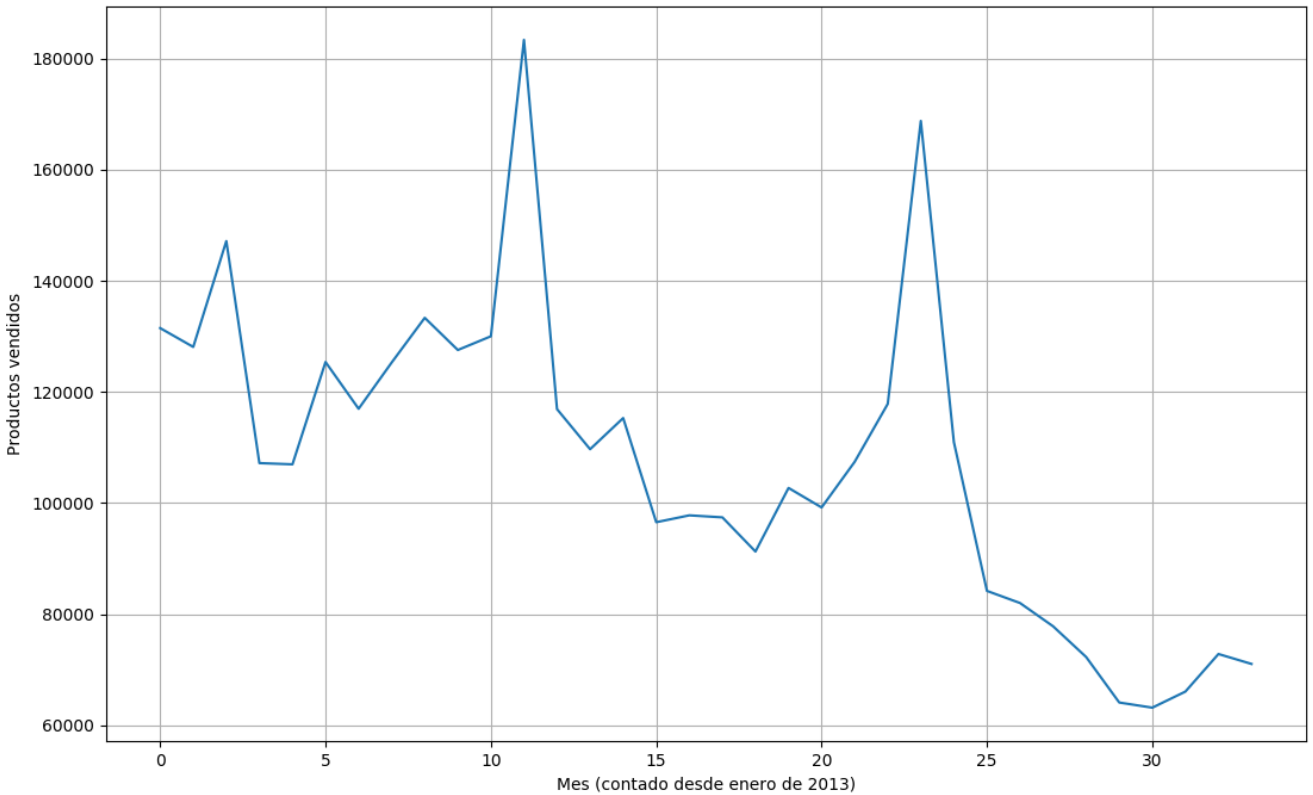
Data information: Original data

- Original data (example):

	date	date_block_num	shop_id	item_id	item_price	item_cnt_day
0	02.01.2013	0	59	22154	999.00	1.0
1	03.01.2013	0	25	2552	899.00	1.0
2	05.01.2013	0	25	2552	899.00	-1.0
3	06.01.2013	0	25	2554	1709.05	1.0
4	15.01.2013	0	25	2555	1099.00	1.0

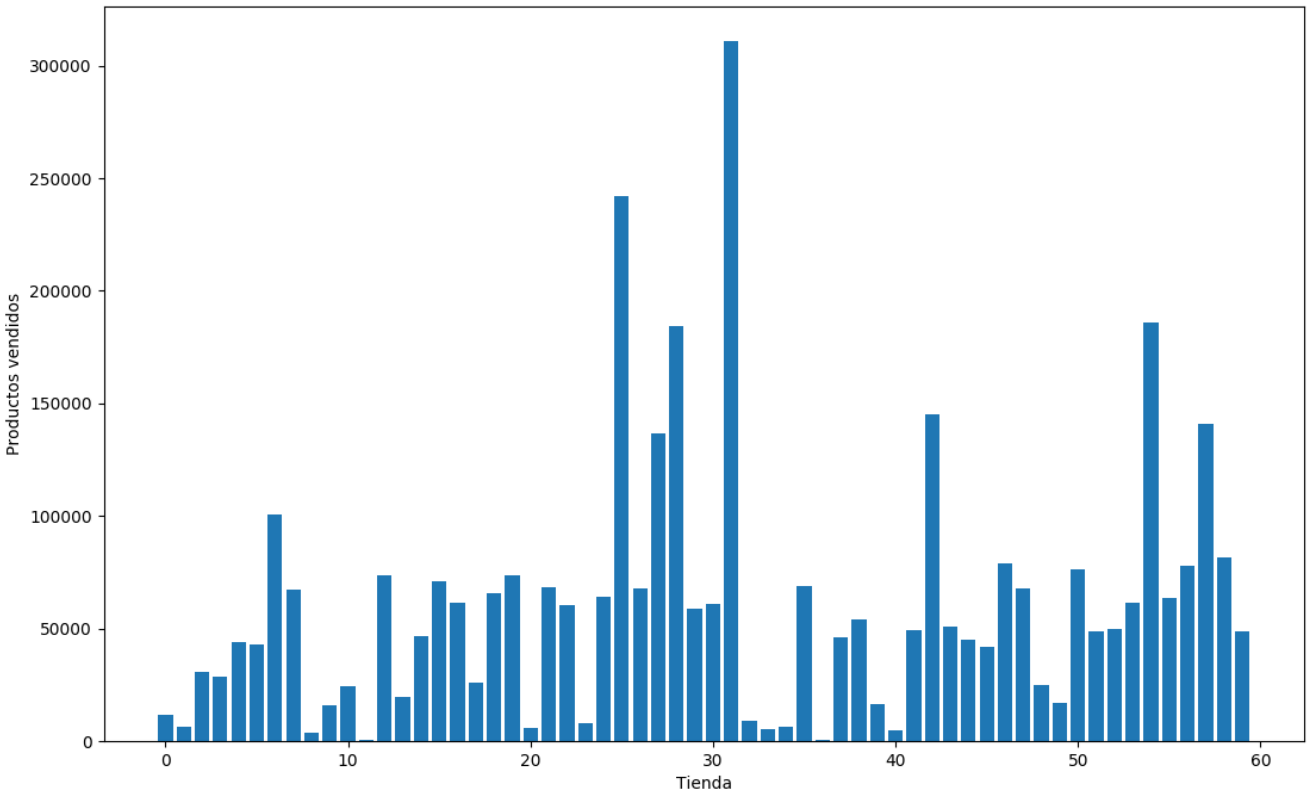
Data information: Total sales

- Total sales:



Data information: Sales by store

- Total sales:



Recurrent networks

1. Theory
   - Recurrent cell
   - Long short-term memory cell
2. Context
   - Problem statement
   - Data information
3. Implementation using TensorFlow
   - Code
   - Alternative Results
   - Results using RNN

Code

- Construction of the network:

```
#LSTM cells
lstm_cells = [tf.contrib.rnn.BasicLSTMCell(num_units=n_neurons, activation=tf.nn.relu) for layer in range(n_layers)]
#Multilayer
multi_cell = tf.contrib.rnn.MultiRNNCell(lstm_cells)
#RNN grouping
outputs, states = tf.nn.dynamic_rnn(multi_cell, X, dtype=tf.float32)
top_layer_h_state = states[-1][1]
#Dense layer
y_pred = tf.layers.dense(top_layer_h_state, n_outputs)
#Loss function
loss = tf.reduce_mean(tf.losses.mean_squared_error(y,y_pred))
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate)
training_op = optimizer.minimize(loss)
```

Code

- TensorFlow Session:

```
with tf.Session() as sess:
    init.run()
    for i in range(len(dataNP[0,:,0])-n_steps):
        x_train = []
        #Reorganizacion del x de entrenamiento para que quede de la forma de X, igual con y_train
        for item in set(sales.item_id):
            x_train.append(X_train[:,i:i+n_steps,item].T)
        x_train = np.array(x_train)
        y_train = dataNP[:,i+n_steps,:].T
        for epoch in range(n_epochs):
                #El batch es un lote con items con los cuales se entrenan las celulas
                indices=np.arange(len(x_train))
                for batch in range(n_batches):
                        X_batch=x_train[indices[batch*batch_size:(batch+1)*batch_size]]
                        y_batch=y_train[indices[batch*batch_size:(batch+1)*batch_size]]
                        sess.run(training_op, feed_dict={X: X_batch, y: y_batch})
    val = y_pred.eval(feed_dict={X: x_test})
    print(val)
```

Recurrent networks

Alternative Results

- Using a random distribution:

$$RMSE_{aleat} = 10.95473$$

- Using ExtraTreesRegressor:

$$RMSE_{regresssor} = 1.56378$$

Recurrent networks

1. Theory
   - Recurrent cell
   - Long short-term memory cell
2. Context
   - Problem statement
   - Data information
3. Implementation using TensorFlow
   - Code
   - Alternative Results
   - Results using RNN

Results using RNN

- Steps: 33; Epochs: 3; Cells: 100

$$RMSE_{RNN} = 1.44038$$

- Steps: 33; Epochs: 10; Cells: 150

$$RMSE_{RNN} = 1.24610$$

- Steps: 6; Epochs: 1; Cells: 150

$$RMSE_{RNN} = 1.22889$$

- Steps: 12; Epochs: 1; Cells: 250

$$RMSE_{RNN} = 1.22481$$