red neuronal recurrente. series temporales

In [2]:

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

In [3]:

```
pwd
```

Out[3]:

```
'C:\\Users\\SARA'
```

In [4]:

```
cd Downloads
```

```
C:\Users\SARA\Downloads
```

In [6]:

```
leche = pd.read_csv('original.csv',index_col='Month')
```

In [7]:

```
leche.head()
```

Out[7]:

	Milk Production
Month	
1962-01-01 01:00:00	589.0
1962-02-01 01:00:00	561.0
1962-03-01 01:00:00	640.0
1962-04-01 01:00:00	656.0
1962-05-01 01:00:00	727.0

In [8]:

```
leche.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 168 entries, 1962-01-01 01:00:00 to 1975-12-01 01:00:00
Data columns (total 1 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   Milk Production  168 non-null    float64
dtypes: float64(1)
memory usage: 2.6+ KB
```

In [9]:

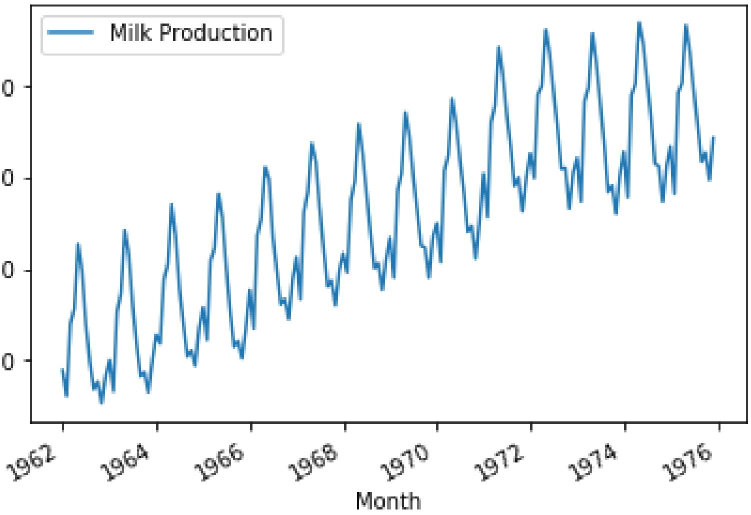
```
leche.index = pd.to_datetime(leche.index)
```

In [10]:

```
leche.plot()
```

Out[10]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1f367851fc8>
```



In [11]:

```
conjunto_entrenamiento = leche.head(150)
conjunto_pruebas = leche.tail(18)
```

In [12]:

```
conjunto_entrenamiento
```

Out[12]:

```
                     Milk Production
Month
1962-01-01 01:00:00            589.0
1962-02-01 01:00:00            561.0
1962-03-01 01:00:00            640.0
1962-04-01 01:00:00            656.0
1962-05-01 01:00:00            727.0
                ...              ...
1974-02-01 01:00:00            778.0
1974-03-01 01:00:00            889.0
1974-04-01 01:00:00            902.0
1974-05-01 01:00:00            969.0
1974-06-01 01:00:00            947.0

150 rows × 1 columns
```

In [13]:

```
conjunto_pruebas
```

Out[13]:

	Milk Production
Month	
1974-07-01 01:00:00	908.0
1974-08-01 01:00:00	867.0
1974-09-01 01:00:00	815.0
1974-10-01 01:00:00	812.0
1974-11-01 01:00:00	773.0
1974-12-01 01:00:00	813.0
1975-01-01 01:00:00	834.0
1975-02-01 01:00:00	782.0
1975-03-01 01:00:00	892.0
1975-04-01 01:00:00	903.0
1975-05-01 01:00:00	966.0
1975-06-01 01:00:00	937.0
1975-07-01 01:00:00	896.0
1975-08-01 01:00:00	858.0
1975-09-01 01:00:00	817.0
1975-10-01 01:00:00	827.0
1975-11-01 01:00:00	797.0
1975-12-01 01:00:00	843.0

In [15]:

```
from sklearn.preprocessing import MinMaxScaler
normalizacion = MinMaxScaler()
entrenamiento_normalizado = normalizacion.fit_transform(conjunto_entrenamiento)
pruebas_normalizado = normalizacion.transform(conjunto_pruebas)
```

In [16]:

```
pruebas_normalizado
```

Out[16]:

```
array([[0.85336538],
       [0.75480769],
       [0.62980769],
       [0.62259615],
       [0.52884615],
       [0.625     ],
       [0.67548077],
       [0.55048077],
       [0.81490385],
       [0.84134615],
       [0.99278846],
       [0.92307692],
       [0.82451923],
       [0.73317308],
       [0.63461538],
       [0.65865385],
       [0.58653846],
       [0.69711538]])
```

In [17]:

```
def lotes(datos_entrenamiento, tamaño_lote, pasos):
    comienzo = np.random.randint(0, len(datos_entrenamiento) - pasos)
    lote_y = np.array(datos_entrenamiento[comienzo:comienzo+pasos+1]).reshape(1,pasos+1)
    return lote_y[:,:-1].reshape(-1,pasos,1), lote_y[:,1:].reshape(-1,pasos,1)
```

In [18]:

```
numero_entradas = 1
numero_pasos = 18
numero_neuronas = 120
numero_salidas = 1
tasa_aprendizaje = 0.001
numero_interaciones_entrenamiento = 5000
tamaño_lote = 1
```

In [20]:

```
x = tf.placeholder(tf.float32, [None, numero_pasos, numero_entradas])
y = tf.placeholder(tf.float32, [None, numero_pasos, numero_salidas])
```

In [22]:

```
capa = tf.contrib.rnn.OutputProjectionWrapper(tf.contrib.rnn.BasicLSTMCell(num_units=numero_neuronas,
activation=tf.nn.relu), output_size=numero_salidas)
```

```
WARNING:tensorflow:
The TensorFlow contrib module will not be included in TensorFlow 2.0.
For more information, please see:
  * https://github.com/tensorflow/community/blob/master/rfcs/20180907-contrib-sunset.md
  * https://github.com/tensorflow/addons
  * https://github.com/tensorflow/io (for I/O related ops)
If you depend on functionality not listed there, please file an issue.

WARNING:tensorflow:From <ipython-input-22-09106b3195e6>:1: BasicLSTMCell.__init__ (from tensorflow.pyt
hon.ops.rnn_cell_impl) is deprecated and will be removed in a future version.
Instructions for updating:
This class is equivalent as tf.keras.layers.LSTMCell, and will be replaced by that in Tensorflow 2.0.
```

In [25]:

```
salidas, estados = tf.nn.dynamic_rnn(capa, x, dtype=tf.float32)
```

```
WARNING:tensorflow:From <ipython-input-25-cb7ea36aed52>:1: dynamic_rnn (from tensorflow.python.ops.rnn
) is deprecated and will be removed in a future version.
Instructions for updating:
Please use `keras.layers.RNN(cell)`, which is equivalent to this API
WARNING:tensorflow:From C:\Users\SARA\anaconda3\envs\pruebasTensorflow\lib\site-packages\tensorflow_c
ore\python\ops\rnn_cell_impl.py:735: Layer.add_variable (from tensorflow.python.keras.engine.base_layer) is d
eprecated and will be removed in a future version.
Instructions for updating:
Please use `layer.add_weight` method instead.
WARNING:tensorflow:From C:\Users\SARA\anaconda3\envs\pruebasTensorflow\lib\site-packages\tensorflow_c
ore\python\ops\rnn_cell_impl.py:739: calling Zeros.__init__ (from tensorflow.python.ops.init_ops) with dtype is
deprecated and will be removed in a future version.
Instructions for updating:
Call initializer instance with the dtype argument instead of passing it to the constructor
```

```
WARNING:tensorflow:From C:\Users\SARA\anaconda3\envs\pruebasTensorflow\lib\site-packages\tensorflow_c
ore\contrib\rnn\python\ops\core_rnn_cell.py:104: calling Constant.__init__ (from tensorflow.python.ops.init_ops
) with dtype is deprecated and will be removed in a future version.
Instructions for updating:
Call initializer instance with the dtype argument instead of passing it to the constructor
```

In [29]:

```
funcion_error = tf.reduce_mean(tf.square(salidas-y))
optimizador = tf.train.AdamOptimizer(learning_rate=tasa_aprendizaje)
entrenamiento = optimizador.minimize(funcion_error)
```

In [36]:

```
init = tf.global_variables_initializer()
saver = tf.train.Saver()
```

In [ ]:

```
with tf.Session() as sesion:
    sesion.run(init)
    for iteracion in range(numero_interaciones_entrenamiento):
        lote_x, lote_y = lotes(entrenamiento_normalizado, tamaño_lote, numero_pasos)
        sesion.run(entrenamiento, feed_dict={x:lote_x, y:lote_y})
        if iteracion %100 == 0:
            error = funcion_error.eval(feed_dict={x:lote_x, y:lote_y})
            print(iteracion, "\t Error ", error)

        saver.save(sesion, "./modelo_series_temporales")
```

```
0    Error  0.33034793
100    Error  0.039746553
200    Error  0.056806043
300    Error  0.013076048
400    Error  0.008069343
500    Error  0.012311291
600    Error  0.009009735
700    Error  0.01185267
800    Error  0.0070071113
900    Error  0.0065021026
1000    Error  0.005237412
1100    Error  0.0073427376
1200    Error  0.008048352
```

In [ ]:

In [ ]:

In [ ]:

In [ ]: