Time Series Exercise -

Follow along with the instructions in bold. Watch the solutions video if you get stuck!

The Data

Source: https://datamarket.com/data/set/22ox/monthly-milk-production-pounds-per-cow-jan-62-dec-75#!ds=22ox&display=line

Use the included .csv file for this exercise. (titled: monthly-milk-production.csv)

Monthly milk production: pounds per cow. Jan 62 - Dec 75

Import numpy pandas and matplotlib

In [1]:

```

```

Use pandas to read the csv of the monthly-milk-production.csv file and set index_col='Month'

In [2]:

```

```

Check out the head of the dataframe

In [3]:

```

```

Out[3]:

	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

Make the index a time series by using:

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

In [4]:

```

```

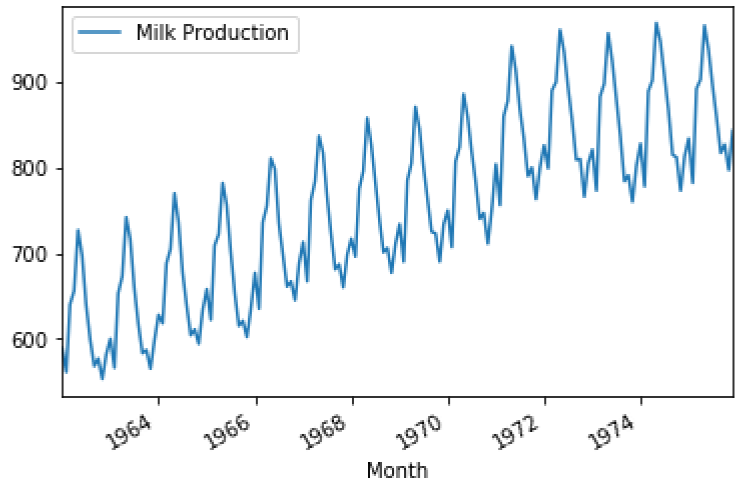
Plot out the time series data.

In [5]:

```

```

Out[5]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1357fc4ec88>
```



Train Test Split

Let's attempt to predict a year's worth of data. (12 months or 12 steps into the future)

Create a test train split using indexing (hint: use .head() or tail() or .iloc[]). We don't want a random train test split, we want to specify that the test set is the last 12 months of data is the test set, with everything before it is the training.

In [40]:

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 168 entries, 1962-01-01 01:00:00 to 1975-12-01 01:00:00
Data columns (total 1 columns):
Milk Production    168 non-null float64
dtypes: float64(1)
memory usage: 2.6 KB
```

In [41]:

In [42]:

Scale the Data

Use sklearn.preprocessing to scale the data using the MinMaxScaler. Remember to only fit_transform on the training data, then transform the test data. You shouldn't fit on the test data as well, otherwise you are assuming you would know about future behavior!

In [9]:

In [10]:

In [11]:

In [12]:

Batch Function

We'll need a function that can feed batches of the training data. We'll need to do several things that are listed out as steps in the comments of the function. Remember to reference the previous batch method from the lecture for hints. Try to fill out the function template below, this is a pretty hard step, so feel free to reference the solutions!

In [13]:

```
def next_batch(training_data,batch_size,steps):
    """
    INPUT: Data, Batch Size, Time Steps per batch
    OUTPUT: A tuple of y time series results. y[:,:-1] and y[:,1:]
    """

    # STEP 1: Use np.random.randint to set a random starting point index for the batch.
    # Remember that each batch needs have the same number of steps in it.
    # This means you should limit the starting point to len(data)-steps

    # STEP 2: Now that you have a starting index you'll need to index the data from
    # the random start to random start + steps + 1. Then reshape this data to be (1,ste

    # STEP 3: Return the batches. You'll have two batches to return y[:,:-1] and y[:,1:
    # You'll need to reshape these into tensors for the RNN to .reshape(-1,steps,1)
```

In [14]:

Setting Up The RNN Model

Import TensorFlow

In [15]:

The Constants

Define the constants in a single cell. You'll need the following (in parenthesis are the values I used in my solution, but you can play with some of these):

- Number of Inputs (1)
- Number of Time Steps (12)
- Number of Neurons per Layer (100)
- Number of Outputs (1)
- Learning Rate (0.03)
- Number of Iterations for Training (4000)

- Batch Size (1)

In [31]:

```

```

Create Placeholders for X and y. (You can change the variable names if you want). The shape for these placeholders should be [None,num_time_steps-1,num_inputs] and [None, num_time_steps-1, num_outputs] The reason we use num_time_steps-1 is because each of these will be one step shorter than the original time steps size, because we are training the RNN network to predict one point into the future based on the input sequence.

In [17]:

```

```

Now create the RNN Layer, you have complete freedom over this, use tf.contrib.rnn and choose anything you want, OutputProjectionWrappers, BasicRNNCells, BasicLSTMCells, MultiRNNCell, GRUCell etc... Keep in mind not every combination will work well! (If in doubt, the solutions used an Outputprojection Wrapper around a basic LSTM cell with relu activation.

In [18]:

```

```

Now pass in the cells variable into tf.nn.dynamic_rnn, along with your first placeholder (X)

In [19]:

```

```

Loss Function and Optimizer

Create a Mean Squared Error Loss Function and use it to minimize an AdamOptimizer, remember to pass in your learning rate.

In [20]:

```

```

Initialize the global variables

In [21]:

```

```

Create an instance of tf.train.Saver()

In [22]:

```

```

Session

Run a tf.Session that trains on the batches created by your next_batch function. Also add an a loss evaluation for every 100 training iterations. Remember to save your model after you are done training.

In [32]:

```

```

In [33]:

```
with tf.Session() as sess:
    # CODE HERE!

    # Save Model for Later
    saver.save(sess, "./ex_time_series_model")
```

```
0       MSE: 0.0628359
100     MSE: 0.00854151
200     MSE: 0.00699567
300     MSE: 0.0156167
400     MSE: 0.00777238
500     MSE: 0.00864684
600     MSE: 0.0159645
700     MSE: 0.00656524
800     MSE: 0.0076439
900     MSE: 0.006401
1000    MSE: 0.00369383
1100    MSE: 0.00988994
1200    MSE: 0.00803645
1300    MSE: 0.00575964
1400    MSE: 0.0151093
1500    MSE: 0.00752775
1600    MSE: 0.00542804
1700    MSE: 0.00162975
1800    MSE: 0.00230503
1900    MSE: 0.00416592
2000    MSE: 0.00369024
2100    MSE: 0.00397327
2200    MSE: 0.00235241
2300    MSE: 0.00472639
2400    MSE: 0.00418429
2500    MSE: 0.00693244
2600    MSE: 0.00375631
2700    MSE: 0.00236074
2800    MSE: 0.00268888
2900    MSE: 0.00708326
3000    MSE: 0.00418036
3100    MSE: 0.00486205
3200    MSE: 0.00659863
3300    MSE: 0.00621194
3400    MSE: 0.00150676
3500    MSE: 0.0050875
3600    MSE: 0.00395521
3700    MSE: 0.00200348
3800    MSE: 0.00386259
3900    MSE: 0.00360108
```

Predicting Future (Test Data)

Show the test_set (the last 12 months of your original complete data set)

In [ ]:

```
# CODE HERE
```

Now we want to attempt to predict these 12 months of data, using only the training data we had. To do this we will feed in a seed training_instance of the last 12 months of the

training_set of data to predict 12 months into the future. Then we will be able to compare our generated 12 months to our actual true historical values from the test set!

Generative Session

NOTE: Recall that our model is really only trained to predict 1 time step ahead, asking it to generate 12 steps is a big ask, and technically not what it was trained to do! Think of this more as generating new values based off some previous pattern, rather than trying to directly predict the future. You would need to go back to the original model and train the model to predict 12 time steps ahead to really get a higher accuracy on the test data. (Which has its limits due to the smaller size of our data set)

Fill out the session code below to generate 12 months of data based off the last 12 months of data from the training set. The hardest part about this is adjusting the arrays with their shapes and sizes. Reference the lecture for hints.

In [43]:

```
with tf.Session() as sess:

    # Use your Saver instance to restore your saved rnn time series model
    saver.restore(sess, "./ex_time_series_model")

    # CODE HERE!
```

```
INFO:tensorflow:Restoring parameters from ./ex_time_series_model
```

Show the result of the predictions.

In [45]:

```
```

Out[45]:
```
[array([ 0.66105769]),
 array([ 0.54086538]),
 array([ 0.80769231]),
 array([ 0.83894231]),
 array([ 1.]),
 array([ 0.94711538]),
 array([ 0.85336538]),
 array([ 0.75480769]),
 array([ 0.62980769]),
 array([ 0.62259615]),
 array([ 0.52884615]),
 array([ 0.625]),
 0.65501654,
 0.60958958,
 0.82095361,
 0.82965684,
 0.97597635,
 0.91560352,
 0.85447896,
 0.78555191,
 0.69337928,
 0.70481831,
 0.64406919,
 0.71613598]
```

Grab the portion of the results that are the generated values and apply inverse_transform on them to turn them back into milk production value units (lbs per cow). Also reshape the results to be (12,1) so we can easily add them to the test_set dataframe.

In [46]:

Create a new column on the test_set called "Generated" and set it equal to the generated results. You may get a warning about this, feel free to ignore it.

In [49]:

```
C:\Users\Marcial\Anaconda3\envs\tf_1_3\lib\site-packages\ipykernel_launcher.py:1: Settin
gWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexi
ng.html#indexing-view-versus-copy
  """Entry point for launching an IPython kernel.
```

View the test_set dataframe.

In [51]:

Out[51]:

	Milk Production	Generated
Month		
1975-01-01 01:00:00	834.0	825.486877
1975-02-01 01:00:00	782.0	806.589233
1975-03-01 01:00:00	892.0	894.516663

	Milk Production	Generated
Month		
1975-04-01 01:00:00	903.0	898.137207
1975-05-01 01:00:00	966.0	959.006165
1975-06-01 01:00:00	937.0	933.891113
1975-07-01 01:00:00	896.0	908.463257
1975-08-01 01:00:00	858.0	879.789612
1975-09-01 01:00:00	817.0	841.445801
1975-10-01 01:00:00	827.0	846.204346
1975-11-01 01:00:00	797.0	820.932739
1975-12-01 01:00:00	843.0	850.912537

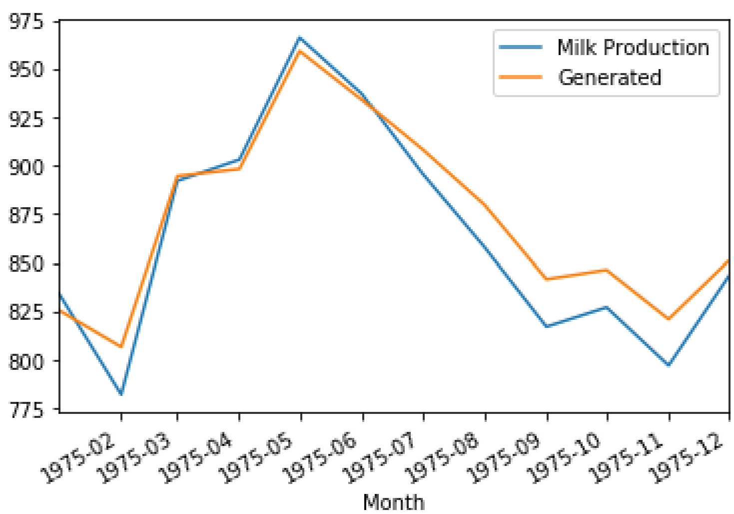
Plot out the two columns for comparison.

In [39]:

```

```

Out[39]: <matplotlib.axes._subplots.AxesSubplot at 0x1377c5ddc18>



Great Job!

Play around with the parameters and RNN layers, does a faster learning rate with more steps improve the model? What about GRU or BasicRNN units? What if you train the original model to not just predict one timestep ahead into the future, but 3 instead? Lots of stuff to add on here!