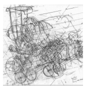


About

MAY 15, 2016 — 14 MIN READ

Sequence prediction using recurrent neural networks(LSTM) with TensorFlow

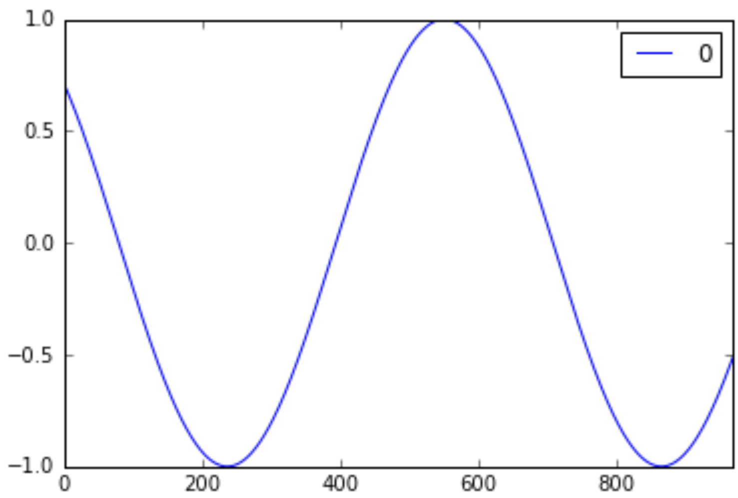
LSTM regression using TensorFlow.

This post tries to demonstrates how to approximate a sequence of vectors using a recurrent neural networks, in particular I will be using the LSTM architecture, The complete code used for this post could be found here. Most of the examples I found in the internet apply the LSTM architecture to natural language processing problems, and I couldn't find an example where this architecture could be used to predict continuous values.

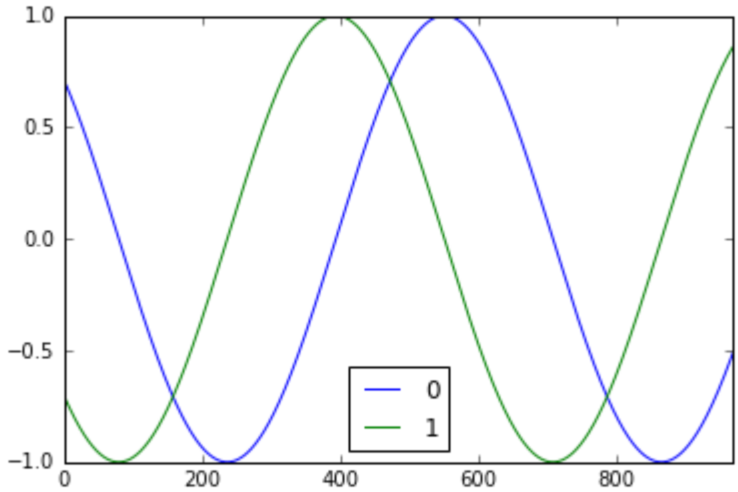
So the task here is to predict a sequence of real numbers based on previous observations. The traditional neural networks architectures can't do this, this is why recurrent neural networks were made to address this issue, as they allow to store previous information to predict future event.

In this example we will try to predict a couple of functions:

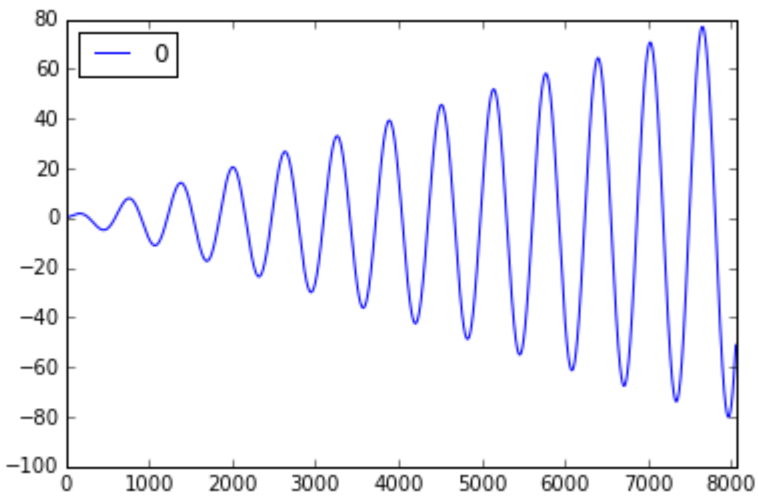
- sin



- sin and cos on the same time



- x*sin(x)



First of all let's build our model, `lstm_model`, the model is a list of stacked lstm cells of different time steps followed by a dense layers.

```
def lstm_model(time_steps, rnn_layers, dense_layers=None):
    def lstm_cells(layers):
        if isinstance(layers[0], dict):
            return [tf.nn.rnn_cell.DropoutWrapper(tf.nn.rnn_cell.BasicLSTM
                    if layer.get('keep_prob') else tf.nn.rnn_cell.BasicLS
                    for layer in layers]
        return [tf.nn.rnn_cell.BasicLSTMCell(steps) for steps in layers]

    def dnn_layers(input_layers, layers):
        if layers and isinstance(layers, dict):
```

```
            return skflow.ops.dnn(input_layers,
                                  layers['layers'],
                                  activation=layers.get('activation'),
                                  dropout=layers.get('dropout'))
        elif layers:
            return skflow.ops.dnn(input_layers, layers)
        else:
            return input_layers

    def _lstm_model(X, y):
        stacked_lstm = tf.nn.rnn_cell.MultiRNNCell(lstm_cells(rnn_layers)
        x_ = skflow.ops.split_squeeze(1, time_steps, X)
        output, layers = tf.nn.rnn(stacked_lstm, x_, dtype=dtypes.float32
        output = dnn_layers(output[-1], dense_layers)
        return skflow.models.linear_regression(output, y)

    return _lstm_model
```

So our model expects a data with dimension corresponding to `(batch size, time_steps of the first lstm cell, num_features in our data)`.

Next we need to prepare the data in a way that could be accepted by our model.

```
def rnn_data(data, time_steps, labels=False):
    """
    creates new data frame based on previous observation
      * example:
        l = [1, 2, 3, 4, 5]
        time_steps = 2
        -> labels == False [[1, 2], [2, 3], [3, 4]]
        -> labels == True [2, 3, 4, 5]
    """
    rnn_df = []
    for i in range(len(data) - time_steps):
        if labels:
            try:
                rnn_df.append(data.iloc[i + time_steps].as_matrix())
            except AttributeError:
                rnn_df.append(data.iloc[i + time_steps])
        else:
            data_ = data.iloc[i: i + time_steps].as_matrix()
            rnn_df.append(data_ if len(data_.shape) > 1 else [[i] for i in
    return np.array(rnn_df)


def split_data(data, val_size=0.1, test_size=0.1):
    """
    splits data to training, validation and testing parts
```

```
    """
    ntest = int(round(len(data) * (1 - test_size)))
    nval = int(round(len(data.iloc[:ntest]) * (1 - val_size)))

    df_train, df_val, df_test = data.iloc[:nval], data.iloc[nval:ntest], d

    return df_train, df_val, df_test


def prepare_data(data, time_steps, labels=False, val_size=0.1, test_size=(
    """
    Given the number of `time_steps` and some data.
    prepares training, validation and test data for an lstm cell.
    """
    df_train, df_val, df_test = split_data(data, val_size, test_size)
    return (rnn_data(df_train, time_steps, labels=labels),
            rnn_data(df_val, time_steps, labels=labels),
            rnn_data(df_test, time_steps, labels=labels))


def generate_data(fct, x, time_steps, seperate=False):
    """generate data with based on a function fct"""
    data = fct(x)
    if not isinstance(data, pd.DataFrame):
        data = pd.DataFrame(data)
    train_x, val_x, test_x = prepare_data(data['a'] if seperate else data,
    train_y, val_y, test_y = prepare_data(data['b'] if seperate else data,
    return dict(train=train_x, val=val_x, test=test_x), dict(train=train_y
```

this will create a data that will allow our model to look `time_steps` number of times back in the past in order to make a prediction. So if for example our first cell is a 10 `time_steps` cell, then for each prediction we want to make, we need to feed the cell 10 historical data points. The `y` values should correspond to the tenth value of the data we want to predict.

Now we can create a regressor based on our our model

```
regressor = skflow.TensorFlowEstimator(model_fn=lstm_model(TIMESTEPS, RNN_L
                                       n_classes=0,
                                       verbose=1,
                                       steps=TRAINING_STEPS,
                                       optimizer='Adagrad',
                                       learning_rate=0.03,
                                       batch_size=BATCH_SIZE)
```

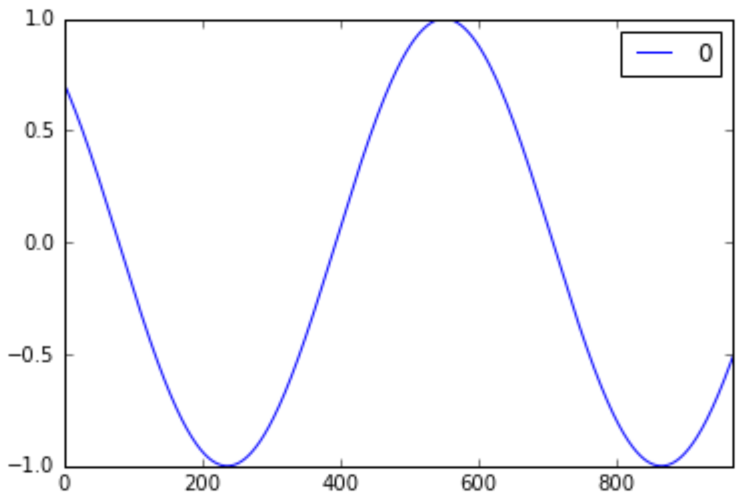
Predicting the `sin` function

```
X, y = generate_data(np.sin, np.linspace(0, 100, 10000), TIMESTEPS, sepera
# create a lstm instance and validation monitor
validation_monitor = skflow.monitors.ValidationMonitor(X['val'], y['val'],
                                                        print_steps=PRINT_S
                                                        early_stopping_rour
                                                        logdir=LOG_DIR)
regressor.fit(X['train'], y['train'], validation_monitor, logdir=LOG_DIR)

# > last training steps
# Step #9700, epoch #119, avg. train loss: 0.00082, avg. val loss: 0.0008
# Step #9800, epoch #120, avg. train loss: 0.00083, avg. val loss: 0.0008
# Step #9900, epoch #122, avg. train loss: 0.00082, avg. val loss: 0.0008
# Step #10000, epoch #123, avg. train loss: 0.00081, avg. val loss: 0.000
```

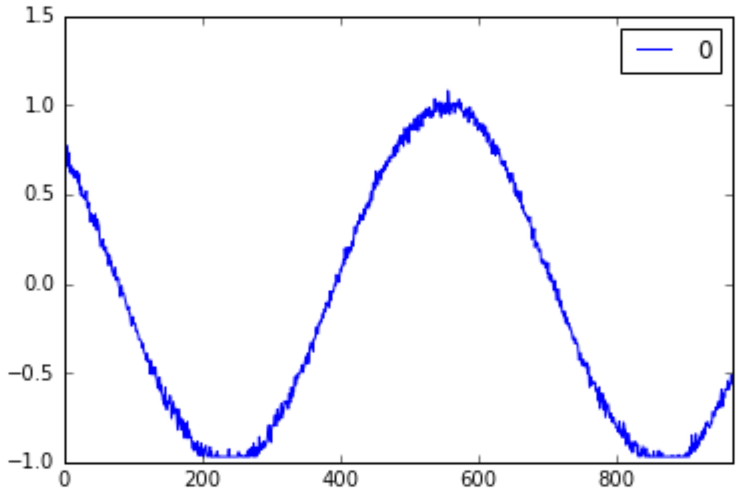
predicting the test data

```
mse = mean_squared_error(regressor.predict(X['test']), y['test'])
print ("Error: {}".format(mse))
# 0.000776
```

- real sin function



- predicted sin function



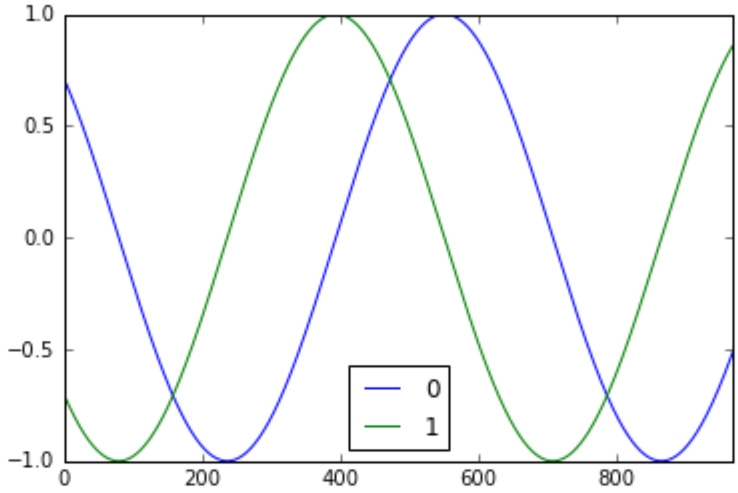
Predicting the `sin and cos` functions together

```
def sin_cos(x):
    return pd.DataFrame(dict(a=np.sin(x), b=np.cos(x)), index=x)

X, y = generate_data(sin_cos, np.linspace(0, 100, 10000), TIMESTEPS, sepe
# create a lstm instance and validation monitor
validation_monitor = skflow.monitors.ValidationMonitor(X['val'], y['val'],
                                                       print_steps=PRINT_S
                                                       early_stopping_rou
                                                       logdir=LOG_DIR)
regressor.fit(X['train'], y['train'], validation_monitor, logdir=LOG_DIR)

# > last training steps
# Step #9500, epoch #117, avg. train loss: 0.00120, avg. val loss: 0.0011
# Step #9600, epoch #118, avg. train loss: 0.00121, avg. val loss: 0.0011
# Step #9700, epoch #119, avg. train loss: 0.00118, avg. val loss: 0.0011
# Step #9800, epoch #120, avg. train loss: 0.00118, avg. val loss: 0.0011
# Step #9900, epoch #122, avg. train loss: 0.00118, avg. val loss: 0.0011
# Step #10000, epoch #123, avg. train loss: 0.00117, avg. val loss: 0.001
```

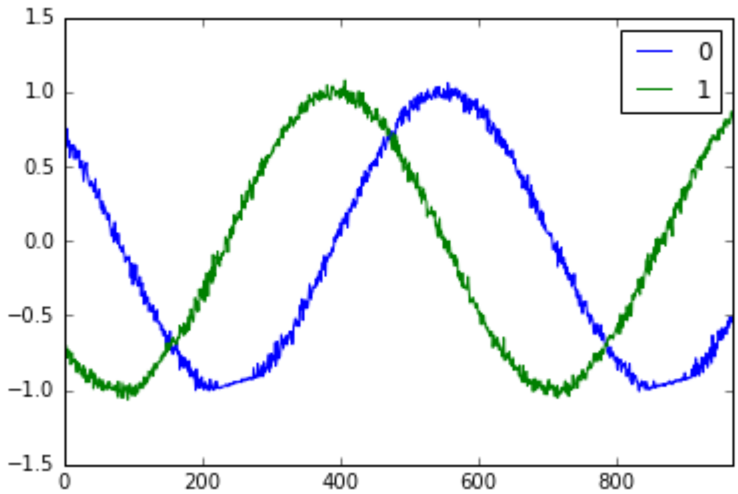
predicting the test data

```
mse = mean_squared_error(regressor.predict(X['test']), y['test'])
print ("Error: {}".format(mse))
# 0.001144
```

- real sin-cos function



- predicted sin-cos function



Predicting the `x*sin` function

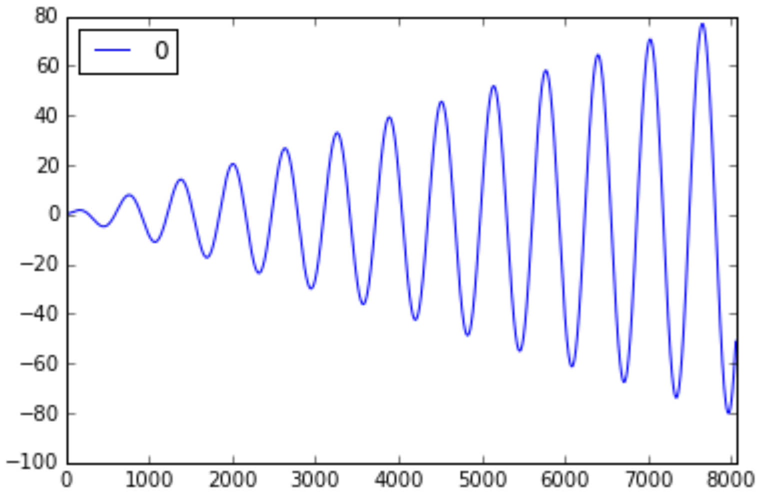
```
def x_sin(x):
    return x * np.sin(x)

X, y = generate_data(x_sin, np.linspace(0, 100, 10000), TIMESTEPS, separa
# create a lstm instance and validation monitor
validation_monitor = skflow.monitors.ValidationMonitor(X['val'], y['val'],
                                                        print_steps=PRINT_S
                                                        early_stopping_rou
                                                        logdir=LOG_DIR)
regressor.fit(X['train'], y['train'], validation_monitor, logdir=LOG_DIR)
```

```

# > last training steps
# Step #32500, epoch #401, avg. train loss: 0.48248, avg. val loss: 15.98(
# Step #33800, epoch #417, avg. train loss: 0.47391, avg. val loss: 15.92!
# Step #35100, epoch #433, avg. train loss: 0.45570, avg. val loss: 15.77:
# Step #36400, epoch #449, avg. train loss: 0.45853, avg. val loss: 15.61(
# Step #37700, epoch #465, avg. train loss: 0.44212, avg. val loss: 15.48(
# Step #39000, epoch #481, avg. train loss: 0.43224, avg. val loss: 15.43!
```

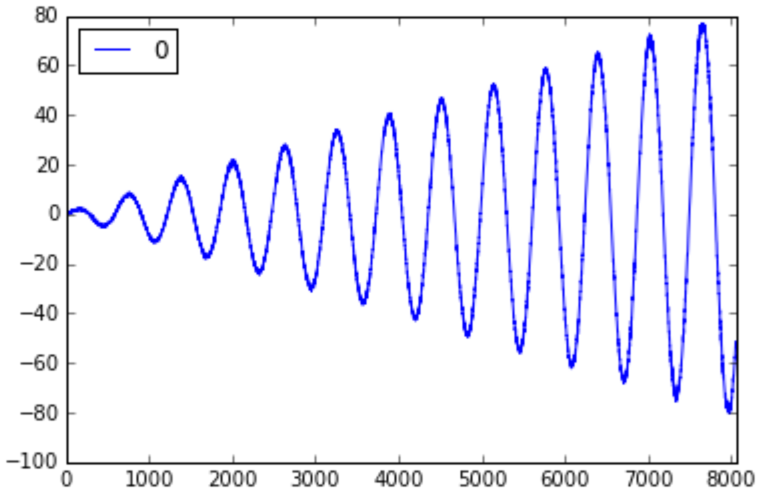
predicting the test data

```
mse = mean_squared_error(regressor.predict(X['test']), y['test'])
print ("Error: {}".format(mse))
# 61.024454351
```

- real x*sin function

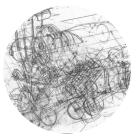


- predicted x*sin function



N.B I am not completely sure if this is the right way to train lstm on regression problems, I am still experimenting with the RNN sequence-to-sequence model, I will update this post or write a new one to use the sequence-to-sequence model.

Tweet Submit 0 reddit this!



*Written by **Mourad Mourafiq.***
Maths, Technology, Philosophy, Startups, ...

Previous story: On infinity

A closer look at the notion of infinity and countable infinity.

2 Comments mourafiq's 1 Login

Recommend 2 Share Sort by Best

Join the discussion…

Hoondy • 3 days ago

Thanks for the great post. It shows MSE for the x*sin function is 61.024454351 and I was wondering what make it so low compare to others.

• Reply • Share ›

mmourafiq Mod → Hoondy • 2 days ago

I had set an `early_stopping_rounds` at 1000, I believe that the model could learn the function better if it was allowed to do more steps, also there are other possibilities to play with, like the initial learning, the number of nodes in the dense and lstm layers, and the number of hidden layers. The objective of the blog post was basically to try the LSTM on continuous values and gather feedback from other people.

• Reply • Share ›

ALSO ON **MOURAFIQ'S**

End to end web app with Django-Rest-Framework &

3 comments • 3 years ago

mmourafiq — Thank you, I posted also a part4 if you are interested.

Quora Answer Classifier. (Part 2, with KNN)

2 comments • 3 years ago

mmourafiq — I wanted to implement the LR, but didn't in the end. For more accurate

End to end web app with Django-Rest-Framework &

7 comments • 3 years ago

Domenico Colandrea — Great tutorial!!! I just read part 1- part 4

End to end web app with Django-Rest-Framework &

14 comments • 3 years ago

mmourafiq — You right about sqlite. I will update the repo.

 Blog · About

Based on Incorporated theme by Inc