Predicting a sine wave

Scripot to predict a sine wave and get practice using LSTM layers in a recurrent neural network.

Sources:

[Soumith's Githib - Time Sequence Prediction]()

```
# Generating Sine Wave Data
------
```

Generating Sine Wave Data

---

[41]
```
import numpy as np
import torch
import matplotlib.pyplot as plt
import cycler
```

Define Parameters for various sine waves

[42]
```
np.random.seed(2)

T = 20
L = 1000
N = 100
```

[43]
```
#Create an empty np matrix (of zeros) with shape 100x1000
x = np.zeros((N, L), 'int64')
print("Shape of matrix x:", x.shape)
print(" ")
print("Top left 5x5 matrix of matrix (x):")
print(x[:5, :5])
```

```
Shape of matrix x: (100, 1000)

Top left 5x5 matrix of matrix (x):
[[0 0 0 0 0]
 [0 0 0 0 0]
 [0 0 0 0 0]
 [0 0 0 0 0]
 [0 0 0 0 0]]
```

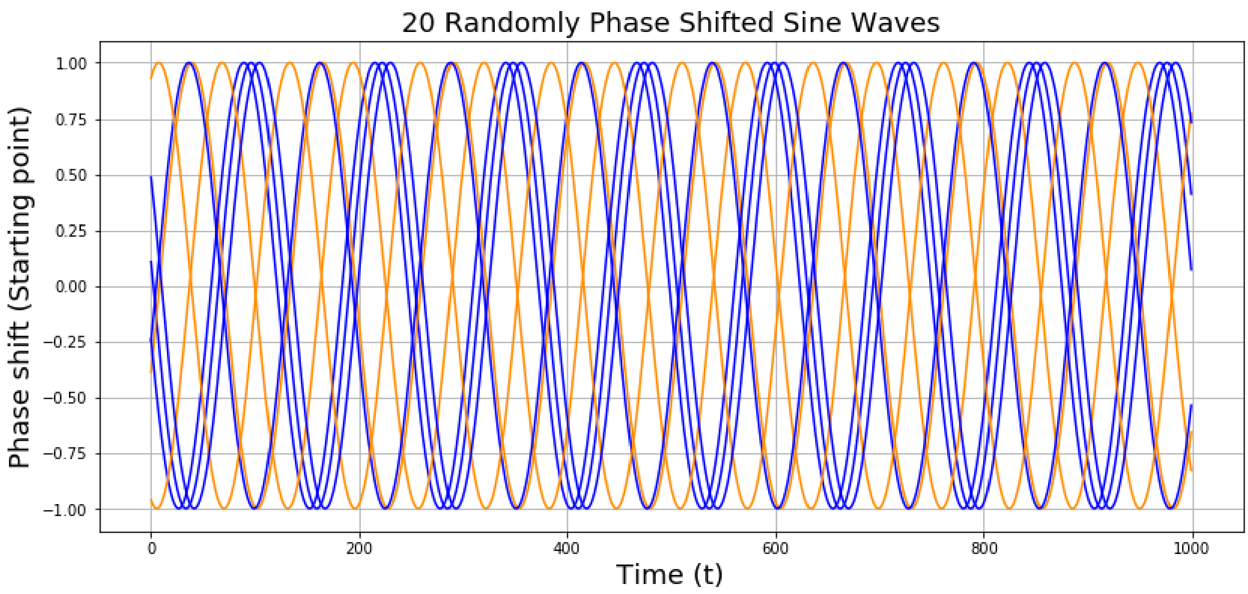
[44]
```
linear_vector = np.array(range(L))
x[:] = linear_vector + np.random.randint(-4 * T, 4 * T, N).reshape(N, 1)
```

These will be the starting points of our 100 phase shifted sine waves

[45]
```
data = np.sin(x / 1.0 / T).astype('float64')
```

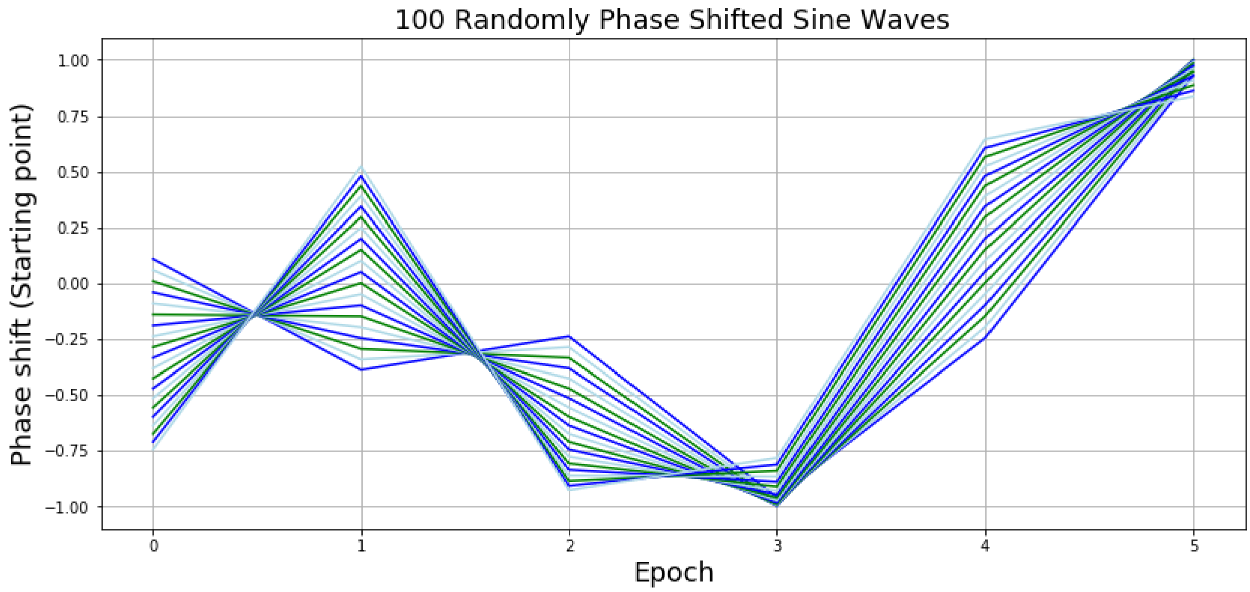
[46]
```
fig = plt.figure(figsize=(30, 6))
ax1 = fig.add_subplot(121)
plt.gca().set_prop_cycle('color', ['blue', 'darkorange'])

plt.plot(data[0, :])
plt.plot(data[1, :])
plt.plot(data[2, :])
plt.plot(data[3, :])
plt.plot(data[4, :])
plt.plot(data[5, :])
plt.plot(data[6, :])
plt.xlabel('Time (t)', fontsize = 18)
plt.ylabel('Phase shift (Starting point)', fontsize = 18)
plt.title('20 Randomly Phase Shifted Sine Waves', fontsize = 18)
plt.grid()
```



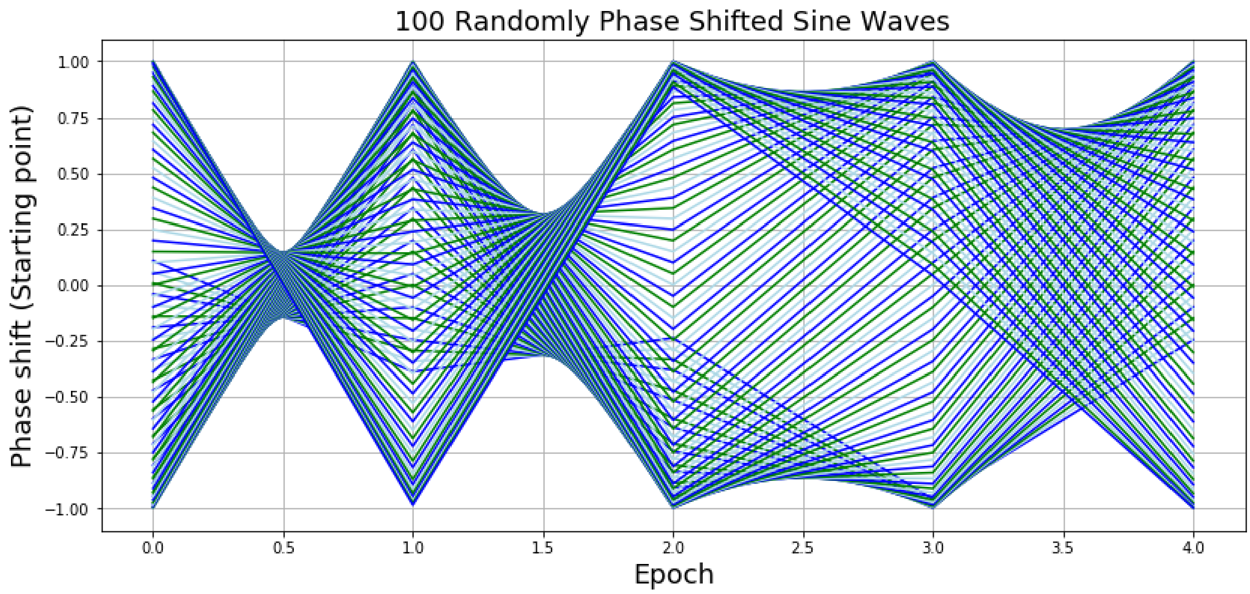
[47]
```
fig = plt.figure(figsize=(30, 6))
ax1 = fig.add_subplot(121)
plt.gca().set_prop_cycle('color', ['blue', 'lightblue', 'green'])

plt.plot(data[:6, :20])
plt.xlabel('Epoch', fontsize = 18)
plt.ylabel('Phase shift (Starting point)', fontsize = 18)
plt.title('100 Randomly Phase Shifted Sine Waves', fontsize = 18)
plt.grid()
```




```
[48] fig = plt.figure(figsize=(30, 6))
     ax1 = fig.add_subplot(121)
     plt.gca().set_prop_cycle('color', ['blue', 'lightblue', 'green'])

     plt.plot(data[:5, :100])
     plt.xlabel('Epoch', fontsize = 18)
     plt.ylabel('Phase shift (Starting point)', fontsize = 18)
     plt.title('100 Randomly Phase Shifted Sine Waves', fontsize = 18)
     plt.grid()
```



Define Network Parameters

Now that we have several randomly phase shifted sine waves stored in 'data' we can begin to build out the network we will use for prediction

```
[49]  from __future__ import print_function
      import torch
      import torch.nn as nn
      import torch.optim as optim
      import numpy as np
      import matplotlib
      matplotlib.use('Agg')
      import matplotlib.pyplot as plt
```

```
/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:7:
UserWarning:
This call to matplotlib.use() has no effect because the backend has already
been chosen; matplotlib.use() must be called *before* pylab,
matplotlib.pyplot,
or matplotlib.backends is imported for the first time.

The backend was *originally* set to
'module://ipykernel.pylab.backend_inline' by the following code:
  File "/anaconda3/lib/python3.6/runpy.py", line 193, in
_run_module_as_main
    "__main__", mod_spec)
  File "/anaconda3/lib/python3.6/runpy.py", line 85, in _run_code
    exec(code, run_globals)
  File "/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py", line
16, in <module>
    app.launch_new_instance()
  File "/anaconda3/lib/python3.6/site-
packages/traitlets/config/application.py", line 658, in launch_instance
    app.start()
  File "/anaconda3/lib/python3.6/site-packages/ipykernel/kernelapp.py",
line 486, in start
    self.io_loop.start()
  File "/anaconda3/lib/python3.6/site-
packages/tornado/platform/asyncio.py", line 127, in start
    self.asyncio_loop.run_forever()
  File "/anaconda3/lib/python3.6/asyncio/base_events.py", line 422, in
run_forever
    self._run_once()
  File "/anaconda3/lib/python3.6/asyncio/base_events.py", line 1432, in
_run_once
    handle._run()
  File "/anaconda3/lib/python3.6/asyncio/events.py", line 145, in _run
```

```
[50]
      class Sequence(nn.Module):
          def __init__(self):
              super(Sequence, self).__init__()
              self.lstm1 = nn.LSTMCell(1, 51)
              self.lstm2 = nn.LSTMCell(51, 51)
              self.linear = nn.Linear(51, 1)

          def forward(self, input, future = 0):
              outputs = []
              h_t = torch.zeros(input.size(0), 51, dtype=torch.double)
              c_t = torch.zeros(input.size(0), 51, dtype=torch.double)
```

```python
        h_t2 = torch.zeros(input.size(0), 51, dtype=torch.double)
        c_t2 = torch.zeros(input.size(0), 51, dtype=torch.double)

        for i, input_t in enumerate(input.chunk(input.size(1), dim=1)):
            h_t, c_t = self.lstm1(input_t, (h_t, c_t))
            h_t2, c_t2 = self.lstm2(h_t, (h_t2, c_t2))
            output = self.linear(h_t2)
            outputs += [output]
        for i in range(future):# if we should predict the future
            h_t, c_t = self.lstm1(output, (h_t, c_t))
            h_t2, c_t2 = self.lstm2(h_t, (h_t2, c_t2))
            output = self.linear(h_t2)
            outputs += [output]
        outputs = torch.stack(outputs, 1).squeeze(2)
        return outputs
```

[51]

```python
# set random seed to 0
np.random.seed(0)
torch.manual_seed(0)

#Define data for training and testing
input = torch.from_numpy(data[3:, :-1])
target = torch.from_numpy(data[3:, 1:])
test_input = torch.from_numpy(data[:3, :-1])
test_target = torch.from_numpy(data[:3, 1:])
```

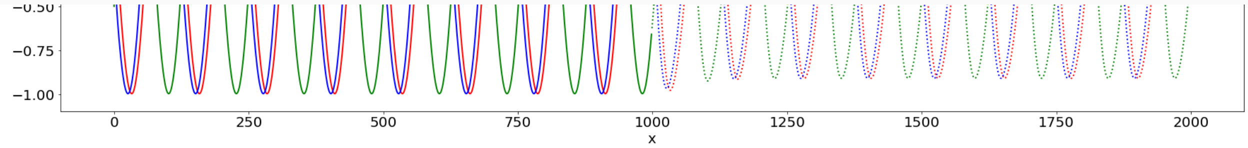
[52]

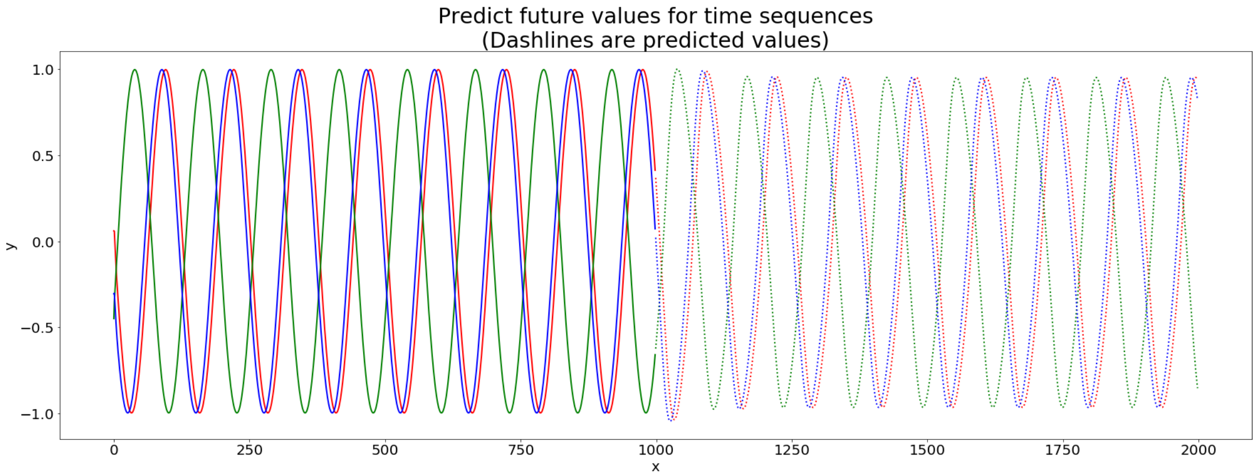
```python
# build the model
seq = Sequence()
seq.double()
criterion = nn.MSELoss()
# use LBFGS as optimizer since we can load the whole data to train
optimizer = optim.LBFGS(seq.parameters(), lr=0.8)
```

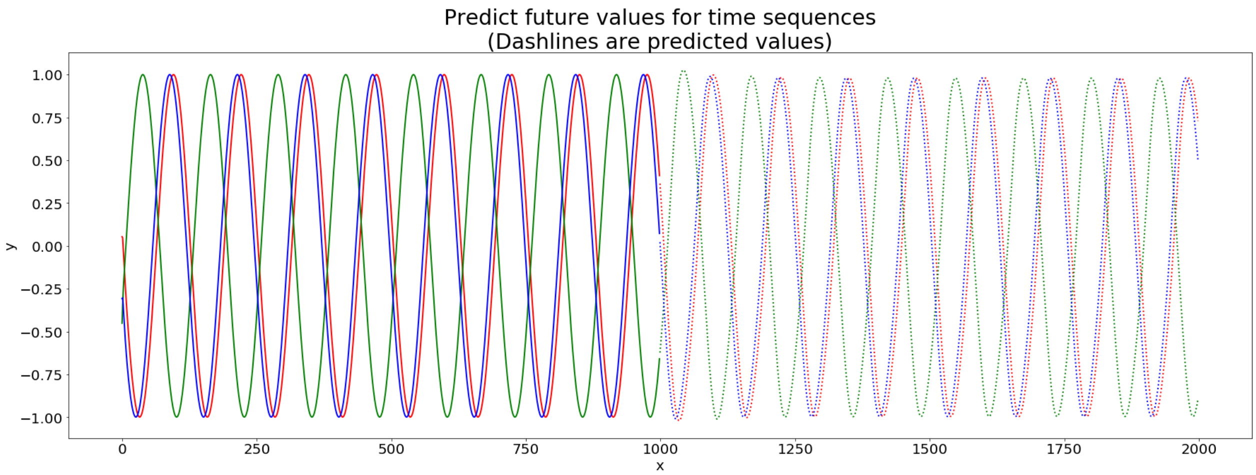
[53]

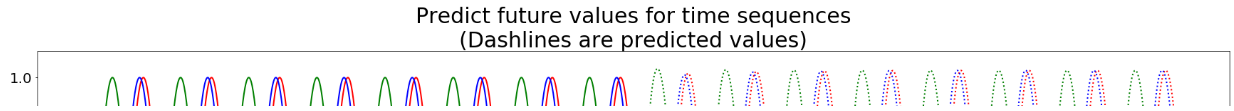
```python
#begin to train
for i in range(15):
    print('STEP: ', i)
    def closure():
        optimizer.zero_grad()
        out = seq(input)
        loss = criterion(out, target)
        print('loss:', loss.item())
        loss.backward()
        return loss
    optimizer.step(closure)
    # begin to predict, no need to track gradient here
```

```python
with torch.no_grad():
    future = 1000
    pred = seq(test_input, future=future)
    loss = criterion(pred[:, :-future], test_target)
    print('test loss:', loss.item())
    y = pred.detach().numpy()
# draw the result
plt.figure(figsize=(30,10))
plt.title('Predict future values for time sequences\n(Dashlines are p
plt.xlabel('x', fontsize=20)
plt.ylabel('y', fontsize=20)
plt.xticks(fontsize=20)
plt.yticks(fontsize=20)
def draw(yi, color):
    plt.plot(np.arange(input.size(1)), yi[:input.size(1)], color, lin
    plt.plot(np.arange(input.size(1), input.size(1) + future), yi[inp
draw(y[0], 'r')
draw(y[1], 'g')
draw(y[2], 'b')
```









We can see that the final iteration produced a result with a test loss of 6*10^-6 and a qualitatively accurate visual continuation of our sine wave.