Implement RNN in PyTorch

(a simple time-series prediction)

Initial Reqs

```
from torch import nn
import numpy as np
import matplotlib.pyplot as plt
```

Stuff for visualizing the data

```
plt.figure(figsize=(8,5))
seq_length = 20          # of data points in each batch
time_steps = np.linspace(0, np.pi, seq_length + 1)
data = np.sin(time_steps)
data.resize((seq_length + 1, 1))     adds a dimension

x = data[:-1]
y = data[1:]

plt.plot(time_steps[1:], x, 'r.', label='input, x')
plt.plot(time_steps[1:], y, 'b.', label='target, y')
plt.legend(loc='best')
plt.show()
```

```python
# Define RNN
class RNN(nn.Module):
    def __init__(self, input_size, output_size,
                 hidden_dim, n_layers):
        super(RNN, self).__init__()
        self.hidden_dim = hidden_dim
```

Actual RNN!

```python
        self.rnn = nn.RNN(input_size, hidden_dim,
                          n_layers, batch_first=True)
        self.fc = nn.Linear(hidden_dim, output_size)
```

batch size X seq-len X input_size

hidden state n_layers X batch_size X hidden_dim

```python
    def forward(self, x, hidden):
        batch_size = x.size(0)
        r_out, hidden = self.rnn(x, hidden)
```

RNN output
batchsize X time_step X hidden_dim

```python
        r_out = r_out.view(-1, self.hidden_dim)
```

new shape:
batch_size * timestep X hidden dim
We are doing some sort of flattening for FC layer.

```python
        output = self.fc(r_out)

        return output, hidden
```

```
# Let's test the RNN!
my_rnn = RNN(input_size=1, output_size=1,
             hidden_dim=10, n_layers=2)

time_steps = np.linspace(0, np.pi, seq_length)
data = np.sin(time_steps)
data.resize((seq_length, 1))
```

add a dim for batch.

```
test_input = torch.Tensor(data).unsqueeze(0)

out, h = my_rnn(test_input, None)

# Training our RNN
input_size = 1
output_size = 1
hidden_dim = 32
n_layers = 1
```

hyperparams

```
rnn = RNN(input_size, output_size,
          hidden_dim, n_layers)
```

Loss & Opt.

```
criterion = nn.MSELoss()
optimizer = torch.optim.Adam(rnn.parameters(), lr=.01)
```

standard for RNNs

Training Loop

```
def train (rnn, n_steps, print_every):
    hidden = None

    for i, step in enumerate(range(n_steps)):
        time_steps = np.linspace (step * np.pi,
                                  (step+1) * np.pi,
                                  seq_length + 1)
        data = np.sin (time_steps)
        data.resize((seq_length +1, 1))

        x = data [:-1]
        y = data [1:]

        x_tensor = torch.Tensor (x). unsqueeze (0)
        y_tensor = torch.Tensor (y)

        prediction, hidden = rnn(x_tensor, hidden)

        # ~~~~~~~~~~~~~~~~~ #
        # ~ representing memory ~ #
        hidden = hidden.data
avoid backpropagating through the entire history

        loss = criterion (prediction, y_tensor)
```

```
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        # Some fancy visualization
        if i % print_every == 0:
            plt.plot(time_steps[1:], x, 'r.')
            plt.plot(time_steps[1:],
                     prediction.data.numpy().flatten(),
                     'b.')
            plt.show()


n_steps = 75
print_every = 15
trained_rnn = train(rnn, n_steps, print_every)
```