Visualization of RNN through Feed-Forward Neural Network

Md Shad Akhtar
Research Scholar
IIT Patna
Problem and Data

Let

- **I/p sequence (X)** : $X^0, X^1, ..., X^T$

- **O/p sequence (O)** : $O^0, O^1, ..., O^T$
Representation of data

Let

- **l/p dimension**: 4
  - $X^0 \rightarrow 0110$

- **o/p dimension**: 3
  - $O^0 \rightarrow 001$
Network Architecture

- Number of neurons at I/p layer : 4
- Number of neurons at O/p layer : 3
- Do we need hidden layers?
  - If yes, number of neurons at each hidden layers
Visualization of RNN through Feed-Forward Neural Network
\[ O^1 = f(W.O^0 + U.X^1) \\
= f([W, U] . [O^0, x^1]) \]
\[ O^2 = f(W.O^1 + U.X^2) = f([W, U] \cdot [O^1, x^2]) \]
$O^{-1}=0$
Different views

View 2

View 1

$O^{-1}=0$

$O=0$

$O=1$

$O=2$

$W=0$

$W=1$

$W=2$

$X=0$

$X=1$

$X=2$

$U$

$U$

$U$

$X$

$X$

$X$

$O$

$O$

$O$

$W$

$W$

$W$

$0=0$

$1=0$

$2=0$
Problem 1: Bit Reverse
Bit Reverse

- Problem definition:
  - **Problem 1**: Reverse a binary digit.
    - 0 → 1 and 1 → 0
  - **Problem 2**: Reverse a sequence of binary digits.
    - 0 1 0 1 0 0 1 → 1 0 1 0 1 1 0
    - Sequence: Fixed or Variable length
  - **Problem 3**: Reverse a sequence of bits over time.
    - 0 1 0 1 0 0 1 → 1 0 1 0 1 1 0
  - **Problem 4**: Reverse a bit if the current i/p and previous o/p are same.

<table>
<thead>
<tr>
<th>Input sequence</th>
<th>Output sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 1 0 0 1 0 0 0 1 1</td>
<td>1 0 1 0 1 0 1 0 1 0</td>
</tr>
</tbody>
</table>
Prepare Data

Let

- **Problem 1**
  - I/p dimension: **1 bit**
  - O/p dimension: **1 bit**

- **Problem 2**
  - Fixed
    - I/p dimension: **10 bit**
    - O/p dimension: **10 bit**
  - Variable: Pad each sequence upto max sequence length: **10**
    - Padding value: **-1**
    - I/p dimension: **10 bit**
    - O/p dimension: **10 bit**

- **Problem 3 & 4**
  - Dimension of each element of I/p (X) : **1 bit**
  - Dimension of each element of O/p (O) : **1 bit**
  - Sequence length : **10**
Network Architecture

Problem 1:
- I/p neurons = 1
- O/p neurons = 1

Problem 2: Fixed & Variable
- I/p neurons = 10
- O/p neurons = 10

Problem 3:
- I/p neurons = 1
- O/p neurons = 1
- Seq len = 10

Problem 4:
- I/p neurons = 1
- O/p neurons = 1
- Seq len = 10

No. of I/p neurons = I/p dimension
No. of O/p neurons = O/p dimension
Implementation using Keras: Basic steps

1. Import necessary libraries
2. Design Network
3. Compile Network
4. Print the the network summary
5. Prepare/Load training data
6. Reshape data \( w.r.t. \) the network
7. Train the network
8. Print final \textit{weights}
9. Evaluate the network
   a. Prepare/Load testing data
   b. Predict o/p
   c. Print test and its prediction
Implementation using Keras: Import necessary libraries

```python
import numpy  # Numpy for mathematical ops
import keras  # Keras main library
from keras.models import Sequential  # Model type
from keras.layers import SimpleRNN  # Recurrent layer
```
Problem 1:
- I/p neurons = 1
- O/p neurons = 1

\[
\begin{align*}
\text{numInNeurons} & = 1 \\
\text{numOutNeurons} & = 1
\end{align*}
\]

\[
\text{model} = \text{Sequential()} \quad \# \text{ Instantiate sequential network}
\]

\[
\text{model.add(Dense(numOutNeurons, input_dim=numInNeurons, activation='sigmoid'))}
\]

Problem 2: Fixed & Variable
- I/p neurons = 10
- O/p neurons = 10

\[
\begin{align*}
\text{numInNeurons} & = 10 \\
\text{numOutNeurons} & = 10
\end{align*}
\]

\[
\text{model.add(Dense(numOutNeurons, input_dim=numInNeurons, activation='sigmoid'))}
\]

\[
\text{# If we need to add more layers we have to call model.add() again. Next time input_dim is not required.}
\]
Implementation using Keras: Design Network

Problem 3:
- I/p neurons = 1
- O/p neurons = 1
- Seq len = 10

\[ U^t = U^{10}, \ldots, U^1, U^0 \]
\[ X^t = X^{10}, \ldots, X^1, X^0 \]
\[ O^t = O^{10}, \ldots, O^1, O^0 \]

\[ \text{numInNeurons} = 1 \]
\[ \text{numOutNeurons} = 1 \]
\[ \text{seqLength} = 10 \]

model = Sequential()  # Instantiate sequential network

# Add a TimeDistributed Dense layer.

model.add(TimeDistributed(Dense(numOutNeurons, activation='sigmoid'), input_shape=(seqLength, numInNeurons)))
Implementation using Keras: Design Network

Problem 4:
- I/p neurons = 1
- O/p neurons = 1
- Seq len = 10

\[
\begin{align*}
O^t & \quad W & \quad O^{t-1} \\
U & \quad W & \\
X^t & \quad U & \\
O^1 & \quad W & \quad W & \quad \ldots & \quad O^{10} \\
W & \quad U & \quad U & \quad W & \quad U & \quad X^{10}
\end{align*}
\]

\[
numInNeurons = 1
\]
\[
numOutNeurons = 1
\]
\[
seqLength = 10
\]

model = Sequential()  # Instantiate sequential network

# Add a SimpleRNN Layer.
model.add(SimpleRNN(numOutNeurons,
    input_shape=(seqLength, numInNeurons),
    return_sequences=True, activation='sigmoid')
model.compile(optimizer='sgd', loss='mse')

# Validate the network. If any issues (dimension mismatch etc.) are found, they will be reported.
# Optimization algorithm is stochastic gradient descent
# Loss is mean squared error

# At this point network is ready for training
### Implementation using Keras: Print the network summary

```python
model.summary()  # Print summary of the network
```

<table>
<thead>
<tr>
<th>Problem</th>
<th>Layer (Type)</th>
<th>Output Shape</th>
<th>Param #</th>
<th>Params Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>dense_1 (Dense)</td>
<td>(None,1)</td>
<td>2</td>
<td>I/p → layer[0] weight : 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>I/p bias weight : 1</td>
</tr>
<tr>
<td>2</td>
<td>dense_1 (Dense)</td>
<td>(None,1)</td>
<td>110</td>
<td>I/p → layer[0] weight : 100</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>I/p bias weight : 10</td>
</tr>
</tbody>
</table>
| 3       | time_distributed_1
(Distributed) | (None,10,1)  | 2       | I/p → layer[0] weight : 1                               |
|         |                                   |              |         | I/p bias weight : 1                                      |
| 4       | simple_rnn_1 (SimpleRNN)          | (None,10,1)  | 3       | I/p → layer[0] weight : 1                               |
|         |                                   |              |         | layer[0] (t-1) → layer[0] (t) weight : 1                |
|         |                                   |              |         | I/p bias weight : 1                                      |
Implementation using Keras: Prepare/Load training data

X = np.loadtxt(open('x.txt','r'))  # load sequence i/p file
O = np.loadtxt(open('o.txt','r'))  # load sequence o/p file

numInstances = 1000  # Number of instances in training data
Implementation using Keras: Reshape data w.r.t. network

- **Problem 1 & 2**
  
  ```python
  X = X.reshape(numInstances, numInNeurons)
  # Input file has ‘numInstances’, each instance has dimension ‘numInNeurons’.
  
  O = O.reshape(numInstances, numOutNeurons)
  # Output file has ‘numInstances’, each instance has dimension ‘numInNeurons’.
  ```

- **Problem 3 & 4**
  
  ```python
  X = X.reshape(numInstances, numUnits, numInNeurons)
  # Input file has ‘numInstances’, each instance has ‘numUnits’ and each unit has dimension ‘numInNeurons’.
  
  O = O.reshape(numInstances, numUnits, numOutNeurons)
  # Output file has ‘numInstances’, each instance has ‘numUnits’ and each unit has dimension ‘numInNeurons’.
  ```
Implementation using Keras:

```python
model.fit(X, O, epochs=5, validation_split=0.2)  # Train network for 5 epochs
```

Epoch 1/5
800/800 [==============================] - 0s - loss: 0.4118 - val_loss: 0.4520

Epoch 2/5
800/800 [==============================] - 0s - loss: 0.4116 - val_loss: 0.4517

Epoch 3/5
800/800 [==============================] - 0s - loss: 0.4114 - val_loss: 0.4514

Epoch 4/5
800/800 [==============================] - 0s - loss: 0.4112 - val_loss: 0.4512

Epoch 5/5
800/800 [==============================] - 0s - loss: 0.4110 - val_loss: 0.4509
Implementation using Keras:  Print final weights

print (model.layers[0].get_weights())  # Print weights of first layer

[  
array([[-0.4387919]], dtype=float32),  # Input to layer[0]
array([[ 0.99820316]], dtype=float32),  # layer[0](t-1) to layer[0](t)
array([-0.00290805], dtype=float32)  # Input bias
]
a. Prepare the test data
   - Problem 1
     test = np.array([[0], [1]])  # Problem 1
   - Problem 2, 3 & 4
     test = np.random.randint(2, size=10)  # Sequence of 1 & 0 of len 10

b. Compute prediction
   - Predict probabilities
     probability = model.predict(test)  # predict o/p probabilities
   - Predict o/p
     prediction = model.predict_classes(test)  # predict o/p sequence
c. Print test, probability and its prediction

```python
print ('Input seq:', test)
print ('Output seq:', prediction)
print ('Probabilities:', probability)
```

<table>
<thead>
<tr>
<th>Problem</th>
<th>Input sequence</th>
<th>Output sequence</th>
<th>Probabilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>[0] [1]</td>
<td>[1] [0]</td>
<td>[0.84571224] [0.11714222]</td>
</tr>
<tr>
<td>2</td>
<td>[1 1 1 1 0 1 1 1 0]</td>
<td>[1 0 0 1 0 0 1 1 0 1]</td>
<td>[0.81408471 0.41276643 0.49722081 0.80422366 0.38091436 0.49808523 0.84009314 0.52011669 0.43402666 0.51361883]</td>
</tr>
<tr>
<td>3</td>
<td>[[1] [0] [1] [0] [1] [1] [0] [0] [1] [0]]</td>
<td>[[0] [1] [0] [1] [0] [1] [0] [0] [1] [0]]</td>
<td>[[0.40637609] [0.50014824] [0.40637609] [0.50014824] [0.40637609] [0.40637609] [0.50014824] [0.50014824] [0.40637609] [0.50014824]]</td>
</tr>
<tr>
<td>4</td>
<td>[[1] [1] [0] [0] [1] [0] [0] [0] [1] [1]]</td>
<td>[[1] [0] [0] [1] [0] [1] [1] [1] [1] [0]]</td>
<td>[[[0.65988613] [0.3504880] [0.46813461] [0.47871199] [0.76111038] [0.55072737] [0.56818104] [0.57156205] [0.57221621] [0.42234275]]</td>
</tr>
</tbody>
</table>
Putting everything together
Implementation using Keras: Problem 1 & 2

```python
# Import libraries
import numpy
import keras
from keras.models import Sequential
from keras.layers import SimpleRNN

numInNeurons = 1  # numInNeurons = 10 for Problem 2
numOutNeurons = 1  # numOutNeurons = 10 for Problem 2
numInstances = 1000

# Design network
model = Sequential()
model.add(Dense(numOutNeurons, input_dim=numInNeurons, activation='sigmoid'))
model.compile(optimizer='sgd', loss='mse')
model.summary()

# Prepare data
X = np.loadtxt(open('x.txt','r'))
O = np.loadtxt(open('o.txt','r'))
X = X.reshape(numInstances, numInNeurons)
Y = Y.reshape(numInstances, numOutNeurons)

# Training
model.fit(X, O, epochs=5, validation_split=0.2)
print (model.layers[0].get_weights())

# Evaluation
test = np.array([[0],[1]])
#test = np.random.randint(2, size=10) for Problem 2
prediction = model.predict_classes(test)
probability = model.predict(test)
print ('Input seq:', test)
print ('Output seq:', prediction)
print ('Probability:', probability)
```

Implementation using Keras: Problem 3

```python
# Import libraries
import numpy
import keras
from keras.models import Sequential
from keras.layers import SimpleRNN

numInNeurons = 1
numOutNeurons = 1
seqLength = 10
numInstances = 1000

# Design network
model = Sequential()
model.add(TimeDistributed(Dense(numOutNeurons, activation='sigmoid'), input_shape=(seqLength, numInNeurons)))
model.compile(optimizer='sgd', loss='mse')
model.summary()

# Prepare data
X = np.loadtxt(open('x.txt','r'))
O = np.loadtxt(open('o.txt','r'))
X = X.reshape(numInstances, seqLength, numInNeurons)
Y = Y.reshape(numInstances, seqLength, numOutNeurons)

# Training
model.fit(X, O, epochs=5, validation_split=0.2)
print (model.layers[0].get_weights())

# Evaluation
test = np.random.randint(2, size=10)
prediction = model.predict_classes(test)
probability = model.predict(test)
print ('Input seq:', test)
print ('Output seq:', prediction)
print ('Probability:', probability)
```
# Import libraries
import numpy
import keras
from keras.models import Sequential
from keras.layers import SimpleRNN

numInNeurons = 1
numOutNeurons = 1
seqLength = 10
numInstances = 1000

# Design network
model = Sequential()
model.add(SimpleRNN(numOutNeurons, input_shape=(seqLength, numInNeurons), return_sequences=True, activation='sigmoid'))
model.compile(optimizer='sgd', loss='mse')
model.summary()

# Prepare data
X = np.loadtxt(open('x.txt','r'))
O = np.loadtxt(open('o.txt','r'))
X = X.reshape(numInstances, seqLength, numInNeurons)
Y = Y.reshape(numInstances, seqLength, numOutNeurons)

# Training
model.fit(X, O, epochs=5, validation_split=0.2)
print (model.layers[0].get_weights())

# Evaluation
test = np.random.randint(2, size=10)
prediction = model.predict_classes(test)
probability = model.predict(test)
print ('Input seq:', test)
print ('Output seq:', prediction)
print ('Probability:', probability)