Unstructured Data Analysis

Lecture 12: Time series analysis with recurrent neural nets

George Chen
Main recurring thing from feedback: challenges in interpreting results. There often isn’t one “best” answer for dimensionality reduction, clustering, topic modeling! Always try to keep in mind: what is the policy question you’re trying to answer, and to what extent do nuances in results matter.
Sequence Data

What we’ve seen so far are “feedforward” NNs
Sequence Data

What we’ve seen so far are “feedforward” NNs

What if we had a video?
Recurrent Neural Nets

Feedforward NN's: treat each video frame separately
Recurrent Neural Nets

Feedforward NN’s: treat each video frame separately

RNNs: feed output at previous time step as input to RNN layer at current time step

In PyTorch, different RNN options: RNN (vanilla), LSTM, GRU
Recurrent Neural Nets

Feedforward NN’s: treat each video frame separately

RNNs: feed output at previous time step as input to RNN layer at current time step

In PyTorch, different RNN options: 
**RNN** (vanilla), **LSTM**, **GRU**
Vanilla ReLU RNN

memory stored in current_state variable!

```python
current_state = np.zeros(num_nodes)
```

```python
for input in input_sequence:
    linear = np.dot(input, W) \  
           + np.dot(current_state, U) \  
           + b

output = np.maximum(0, linear)  # ReLU

current_state = output
```

Parameters: weight matrices $W$ & $U$, and bias vector $b$

Key idea: it’s like a linear layer in a for loop with some memory!
Recurrent Neural Nets

- Like a linear layer that has memory
- Readily chains together with other neural net layers

Feedforward NN’s:
- Treat each video frame separately

RNNs:
- Feed output at previous time step as input to RNN layer at current time step

In PyTorch, different RNN options:
- RNN (vanilla), LSTM, GRU
Recurrent Neural Nets

like a linear layer
that has memory
readily chains together with
other neural net layers

Feedforward NN’s:
treat each video frame separately

RNNs:
feed output at previous
time step as input to
RNN layer at current
time step

In PyTorch, different
RNN options:
RNN (vanilla),
LSTM, GRU

Time series
CNN
RNN layer
like a linear layer
that has memory
Recurrent Neural Nets

- readily chains together with other neural net layers
- like a linear layer that has memory

Feedforward NN’s:
- treat each video frame separately

RNNs:
- feed output at previous time step as input to RNN layer at current time step

In PyTorch, different RNN options:
- RNN (vanilla), LSTM, GRU
Recurrent Neural Nets

Example: Given text (e.g., movie review, Tweet), figure out whether it has positive or negative sentiment (binary classification)

Common first step for text: turn words into vector representations that are semantically meaningful
(Flashback) Do Data Actually Live on Manifolds?

Recurrent Neural Nets

Example: Given text (e.g., movie review, Tweet), figure out whether it has positive or negative sentiment (binary classification)

Common first step for text: turn words into vector representations that are semantically meaningful

In PyTorch, use the Embedding layer
Sentiment Analysis with IMDb Reviews

Step 1: Tokenize & build vocabulary

<table>
<thead>
<tr>
<th>Word index</th>
<th>Word</th>
<th>2D Embedding</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>this</td>
<td>[-0.57, 0.44]</td>
</tr>
<tr>
<td>1</td>
<td>movie</td>
<td>[0.38, 0.15]</td>
</tr>
<tr>
<td>2</td>
<td>rocks</td>
<td>[-0.85, 0.70]</td>
</tr>
<tr>
<td>3</td>
<td>sucks</td>
<td>[-0.26, 0.66]</td>
</tr>
</tbody>
</table>

Step 2: Encode each review as a sequence of word indices into the vocab

“this movie rocks” → 0 1 2
“this movie sucks” → 0 1 3
“this sucks” → 0 3

Step 3: Use word embeddings to represent each word
Sentiment Analysis with IMDb Reviews

Step 1: Tokenize & build vocabulary

<table>
<thead>
<tr>
<th>Word index</th>
<th>Word</th>
<th>2D Embedding</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>this</td>
<td>[-0.57, 0.44]</td>
</tr>
<tr>
<td>1</td>
<td>movie</td>
<td>[0.38, 0.15]</td>
</tr>
<tr>
<td>2</td>
<td>rocks</td>
<td>[-0.85, 0.70]</td>
</tr>
<tr>
<td>3</td>
<td>sucks</td>
<td>[-0.26, 0.66]</td>
</tr>
</tbody>
</table>

Step 2: Encode each review as a sequence of word indices into the vocab

“This movie sucks” → 0 1 3

Step 3: Use word embeddings to represent each word

[-0.57, 0.44]  
[0.38, 0.15]  
[-0.26, 0.66]
Sentiment Analysis with IMDb Reviews

“this movie sucks”

0 1 3

Embedding

[-0.57, 0.44]
[0.38, 0.15]
[-0.26, 0.66]
Sentiment Analysis with IMDb Reviews

“this movie sucks”

0 → Embedding → [-0.57, 0.44]

1 → Embedding → [0.38, 0.15]

3 → Embedding → [-0.26, 0.66]
Sentiment Analysis with IMDb Reviews

“this movie sucks”

Logistic Regression
Sentiment Analysis with IMDb Reviews

RNN’s work with variable-length inputs

Note: Typically in text analysis, the word embeddings are treated as fixed, so we do not update them during training.
Sentiment Analysis with IMDb Reviews

Demo
RNNs: a little bit more detail
(Flashback) Vanilla ReLU RNN

memory stored in current_state variable!

```
current_state = np.zeros(num_nodes)
```

for input in input_sequence:

```
linear = np.dot(input, W) \
    + np.dot(current_state, U) \
    + b
```

```
output = np.maximum(0, linear)  # ReLU
```

```
current_state = output
```

Parameters: weight matrices $W$ & $U$, and bias vector $b$

Key idea: it’s like a linear layer in a for loop with some memory!
(Flashback) Vanilla ReLU RNN

memory stored in current_state variable!

```
current_state = np.zeros(num_nodes)
```

```
outputs = []
```

```
for input in input_sequence:
    linear = np.dot(input, W) \ 
             + np.dot(current_state, U) \ 
             + b

    output = np.maximum(0, linear) # ReLU
    outputs.append(output)

    current_state = output
```
RNN layer

Time series

RNN layer

output prediction
Vanilla RNN tends to forget things quickly.

```python
outputs[t] = np.maximum(np.dot(input_sequence[t], W) + np.dot(outputs[t-1], U) + b, 0)
```
Add explicit long-term memory!

But need some way to update long-term memory!
Time $t-1$

Long-term memory

Add explicit long-term memory!

But need some way to update long-term memory!

Time $t$

output $t-1$

output $t$
Add explicit long-term memory!

But need some way to update long-term memory!

output $t - 1$

output $t$
Add explicit long-term memory!

But need some way to update long-term memory!

Called a “long short-term memory” (LSTM) RNN

Remembers things longer than vanilla RNN
Recurrent Neural Nets

• Neatly handles time series, remembering things over time

• An RNN layer by itself doesn’t take advantage of image/text structure!
  • For images: combine with CNN basic building block (convolutional layer + pooling)
  • For text: combine with embedding layer (use pre-trained word embedding like GloVe, word2vec)
Analyzing Times Series with CNNs

- Think about an image with 1 column, and where the rows index time steps: this is a time series!

- Think about a 2D image where rows index time steps, and the columns index features: this is a multivariate time series (feature vector that changes over time!)

- CNNs can be used to analyze time series but inherently the size of the filters used say how far back in time we look

- If your time series does not have long-range dependencies that require long-term memory, CNNs can do well already!

- If you need long-term memory, use RNNs