Lecture 13: Wrap up CNNs, time series analysis with recurrent neural nets (RNNs)
CNNs

Demo
Time Series Data

Each data point is a video
Previous coverage: MLPs & CNNs can handle each frame separately
Recurrent Neural Nets

Previous coverage: MLPs & CNNs can handle each frame separately

RNNs: include output at previous time step as input to current time step

There are different kinds of RNNs, such as: RNN (vanilla), LSTM, GRU
Recurrent Neural Nets

Previous coverage: MLPs & CNNs can handle each frame separately

RNNs: include output at previous time step as input to current time step

There are different kinds of RNNs, such as: RNN (vanilla), LSTM, GRU
Vanilla ReLU RNN

How memory changes from one time step to the next is determined by an operation that looks like a linear layer followed by a nonlinear activation.

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

$\text{input}$ is a 1D table: $\text{num} \_ \text{features}$ entries

$\text{current} \_ \text{state} = \text{np} \_ \text{zeros}(\text{num} \_ \text{nodes})$

$\text{for} \ \text{input} \ \text{in} \ \text{input} \_ \text{sequence}$:

$\text{linear} = \text{np} \_ \text{dot}(\text{input}, \ W \ _ \ T) + b + \text{np} \_ \text{dot}(\text{current} \_ \text{state}, \ U \ _ \ T)$

$\text{output} = \text{np} \_ \text{maximum}(0, \ \text{linear}) \ # \ \text{ReLU}$

$\text{current} \_ \text{state} = \text{output}$

$\text{linear}$ is a 1D table: $\text{num} \_ \text{nodes}$ entries

$W$ is a 2D table: # rows: $\text{num} \_ \text{nodes}$, # cols: $\text{num} \_ \text{features}$

$b$ is a 1D table: $\text{num} \_ \text{nodes}$ entries

$U$ is a 2D table: $\text{num} \_ \text{nodes}$ by $\text{num} \_ \text{nodes}$

Python list that can have any nonzero length!
Vanilla ReLU RNN

current_state = np.zeros(num_nodes)

outputs = []

for input in input_sequence:
    linear = np.dot(input, W.T) + b  
    + np.dot(current_state, U.T)
    output = np.maximum(0, linear) # ReLU
    outputs.append(output)
    current_state = output

For simplicity, in today’s lecture, we only use the very last time step’s output
Recurrent Neural Nets

Time series → RNN layer

- models how output changes over time but does not know image or text structure!!!

⇒ combine with other neural net layers
Recurrent Neural Nets

apply CNN to each video frame to extract semantically meaningful representation.

Time series

CNN

RNN layer models how output changes over time but does not know image or text structure!!!

⇒ combine with other neural net layers
Conv2d, ReLU → Max Pool 2d → Conv2d, ReLU → Max Pool 2d → Flatten

Actually, intermediate representations close to the last layer are also similar!

(intuition: recall the crumpled paper analogy!)
Recurrent Neural Nets

apply CNN to each video frame to extract semantically meaningful representation

Time series

RNN layer models how output changes over time but does not know image or text structure!!!

⇒ combine with other neural net layers
Recurrent Neural Nets

apply CNN to each video frame to extract semantically meaningful representation

Time series → CNN → RNN layer → Classifier

→ models how output changes over time but *does not know image or text structure***!!!

⇒ combine with other neural net layers
Recurrent Neural Nets

apply CNN to each video frame to extract semantically meaningful representation

Time series → Conv2d, ReLU → Max Pool 2d → Conv2d, ReLU → Max Pool 2d → Flatten → RNN layer → Classifier
Recurrent Neural Nets

apply CNN to each video frame to extract semantically meaningful representation

Time series → CNN → RNN layer → Classifier

RNN layer models how output changes over time but does not know image or text structure!!!

⇒ combine with other neural net layers
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

Text → RNN layer → Classifier → Positive/negative sentiment

Linear layer (2 nodes), Softmax activation
label 0: negative sentiment
label 1: positive sentiment
(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

Linear layer (2 nodes), Softmax activation
label 0: negative sentiment
label 1: positive sentiment
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

Ordering of words matters
Different reviews can have different lengths
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”

Embedding

0 ➔ [-0.57, 0.44]

1 ➔ [0.38, 0.15]

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

Embedding

0 → [-0.57, 0.44]

Embedding

“this movie sucks”

1 → [0.38, 0.15]

Embedding

3 → [-0.26, 0.66]

RNN layer
For this lecture, we only keep the last time step’s output.
Sentiment Analysis with IMDb Reviews

Each “layer” in orange dotted box corresponds to an iteration of the RNN’s for loop & these layers share the same parameters!

“this movie sucks”
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]

Each "layer" in orange dotted box corresponds to an iteration of the RNN's for loop & these layers share the same parameters!
**Sentiment Analysis with IMDb Reviews**

- **Embedding**
  - 0
  - [0.57, 0.44]

- **Embedding**
  - 3
  - [0.26, 0.66]

- **Classifier**

**RNNs work with variable-length inputs**

Each “layer” in orange dotted box corresponds to an iteration of the RNN's for loop & these layers share the same parameters!

Note: Often in text analysis, the word embeddings are treated as fixed, so we do not update them during training.
What if we didn’t use word embeddings?
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]
Bad Strategy: One-Hot Encoding

Step 1: Tokenize & build vocabulary

<table>
<thead>
<tr>
<th>Word index</th>
<th>Word</th>
<th>One-hot encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>this</td>
<td>[1, 0, 0, 0, 0]</td>
</tr>
<tr>
<td>1</td>
<td>movie</td>
<td>[0, 1, 0, 0, 0]</td>
</tr>
<tr>
<td>2</td>
<td>rocks</td>
<td>[0, 0, 1, 0, 0]</td>
</tr>
<tr>
<td>3</td>
<td>sucks</td>
<td>[0, 0, 0, 1, 0]</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 one-hot encoding to represent each word

This strategy tends to work poorly in practice: distance between every pair of words is the same in one-hot encoding!
Recap/Important Reminder

• Neural nets are not doing magic; incorporating structure is very important to state-of-the-art deep learning systems

• Word embeddings encode semantic structure—words with similar meaning are mapped to nearby Euclidean points

• CNNs encode semantic structure for images—images that are “similar” are mapped to nearby Euclidean points

• An RNN tracks how what’s stored in memory changes over time — an RNN’s job is made easier if the memory is a semantically meaningful representation
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

embedding_matrix (100-dimensional GloVe embeddings in the demo)

In the demo, this part done by tokenizing all training data using **spaCy**
Variable-Length Time Series in PyTorch

In PyTorch, how do we specify a batch of time series of varying lengths?

Example: 5 data points (each one is a time series) of lengths 3, 2, 5, 1, 7

Common way: give a 2D table with all time series padded to the max length, and also give a 1D table specifying the lengths.

This shows up in the demo when we specify an example input to the neural net.
Sentiment Analysis with IMDb Reviews

1. Load in training data (25000 IMDb reviews)

2. Do a 80/20 split of the training data into:
   - proper training data (20000 reviews)
   - validation data (5000 reviews)

3. Convert each proper training review into tokens using spaCy
   "Master cinéaste Alain Resnais likes to work with those actors"
   (using the tokenizer function)
   ['master', 'cinéaste', 'alain', 'resnais', 'likes', 'to', 'work', 'with', 'those', 'actors']

4. Build a vocabulary using the proper training reviews
   vocab behaves like a function (input: list of strings, output: list of integers)
1. Load in training data (25000 IMDb reviews)

2. Do a 80/20 split of the training data into:
   - proper training data (20000 reviews)
   - validation data (5000 reviews)

3. Convert each proper training review into tokens using spaCy

4. Build a vocabulary using the proper training reviews

5. Compute each proper training review’s encoded version

"Master cinéaste Alain Resnais likes to work with those actors"

\[
\text{'master', 'cinéaste', 'alain', 'resnais', 'likes', 'to', 'work', 'with', 'those', 'actors'}
\]

\[
[1259, 59266, 11261, 16475, 1225, 7, 171, 20, 162, 169]
\]
6. Construct neural net (instead of `nn.Sequential`, we make a class that inherits from `nn.module`)

PyTorch convention: the `forward` function specifies how a neural net actually processes a batch of input data.

Example: 5 data points (each one is a time series) of lengths: 3, 2, 5, 1, 7.

- Blue entries contain actual values from the 5 time series.
- Gray entries contain padded values (e.g., zeros).

The neural net we constructed has a `forward` function with two inputs:
- A 2D table (each column is for 1 data point).
- A 1D table (specifies length for each time series).

Proper `train_dataset_encoded` is a list of length-2 tuples each containing (encoded review, label 0 or 1).

Val `dataset_encoded`
Example: 5 data points (each one is a time series) of lengths 3, 2, 5, 1, 7.

<table>
<thead>
<tr>
<th>Time steps</th>
<th>Data point</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Blue entries contain actual values from the 5 time series</td>
</tr>
<tr>
<td></td>
<td>Gray entries contain padded values (e.g., zeros)</td>
</tr>
</tbody>
</table>

The neural net we constructed has a forward function with two inputs:
- A 2D table (each column is for 1 data point)
- A 1D table (specifies length for each time series)

```
In [30]:
# example where there are 5 input time series of lengths 3, 2, 5, 1, 7;
# we specify these time series using a 2D table that is padded and a 1D table of lengths (see lecture slides for details)
summary(simple_lstm_model,
       input_data=[torch.zeros((7, 5), dtype=torch.long),
                   torch.tensor([3, 2, 5, 1, 7], dtype=torch.long)])
```

Data types matter in PyTorch (torch.long means these tables store integers)
7. Train the neural net for some user-specified max number of epochs

8. Automatically tune on one hyperparameter: choose # of epochs to be the one achieving highest validation accuracy

9. Load in the saved neural net from the best # of epochs

10. Finally load in test data, tokenize and convert each test review into a list of integers, and use the trained neural net to predict
Sentiment Analysis with IMDb Reviews

Demo