Recurrent Neural Networks
SpeakUp

Join by URL: https://web.speakup.info/room/join/80740

Join by key: web.speakup.info → use key 80740

Join by saved session: web.speakup.info

Join by QR:
Outline

• Recap convolutional neural networks
• Compare conventional and recurrent neural networks (RNNs)
• Text translation example
• Backpropagation through time
• Code comparison
• Parts-of-speech example
• RNN paradigms
• LSTMs
• A mention of attention
Recap: Convolutional Neural Networks

• Take five minutes to draw
Image Dataset

• We need an image dataset for next week’s lecture.
• I want you all to take photos of whatever we’re classifying.

• Candidates so far
  • Big Bridges vs. Little Bridges
  • Frank Food vs. Frary Food
Conventional Neural Networks (including CNNs)

• Input: fixed sized tensor
  • Though the batch size can be any value due to broadcasting

• Output: fixed sized tensor
  • Though the batch size can be any value due to broadcasting

• Functionally deterministic (always produce the same output for a given input)
  • When might you want different outputs on the same input?
Motivating Example: Text Translation

• Input: I love purple cats. Cats are neat.
• Output: J'adore les chats violets. Les chats sont soignés.
Recurrent Neural Networks

Operate over sequences (data with temporal dependencies).

And maintain internal context

\[ X \rightarrow \text{RNN} \rightarrow y \]

\[ X \rightarrow \text{RNN} \rightarrow \text{RNN} \]

Conventional NN

\[ y = f(\theta, x) \]

Recurrent NN

\[ y = f(\theta, x, h) \]
**Linear vs (Elman) Recurrent Neurons**

- **Input shape**: (N, input_size)
- **Output shape**: (N, output_size)

```python
class Neuron(torch.Module):
    def __init__(self, input_size, output_size):
        self.W = torch.randn(output_size, input_size) * 0.01
        self.b = torch.randn(output_size, 1) * 0.01

    def forward(self, X):
        linear = X @ self.W.T + self.b.T
        return F.sigmoid(linear)
```

- **Input shape**: (L, N, input_size)
- **Output shape**: (L, N, output_size)

```python
class RecurrentNeuron():
    def __init__(self, input_size, output_size):
        self.Wx = torch.randn(output_size, input_size) * 0.01
        self.Wh = torch.randn(output_size, output_size) * 0.01
        self.bh = torch.zeros(output_size, 1)
        self.output_size = output_size

    def forward(self, X, state=None):
        L, N, input_size = X.shape
        if not state:
            state = torch.zeros(N, self.output_size)
        output_sequence = []
        for x_t in X:
            state = F.tanh(x_t @ self.Wx + state @ self.Wh + self.bh)
            output_sequence.append(state)
        return torch.tensor(output_sequence), state
```

**Internal State Parameters**

- **Batch Size**
- **Sequence Length**

*Untested code.*
Unrolled Visualization
Motivating Example: Text Translation

- Input: I love purple cats. Cats are neat.
- Output: J'adore les chats violets. Les chats sont soignés.
Backpropagation Through Time (BPTT)

\[ J = \frac{1}{2} (y - y')^2 \]
Backpropagation Through Time (BPTT)
Some BPTT Math

\[ L(y, y) = \frac{1}{2} (y - y)^2 \]
\[ L_2 = \frac{1}{2} (\hat{y}_2 - y_2)^2 \]
\[ \hat{y}_2 = h_2 = x_2 \cdot W_x + h_1 \cdot W_n + b \]

\[ \frac{\partial L_2}{\partial W_x} = \frac{\partial L_2}{\partial \hat{y}_2} \cdot \frac{\partial \hat{y}_2}{\partial W_x} \]
\[ \frac{\partial L_2}{\partial W_n} = \frac{\partial L_2}{\partial \hat{y}_2} \cdot \frac{\partial \hat{y}_2}{\partial W_n} + \frac{\partial \hat{y}_2}{\partial \hat{y}_2} \cdot \frac{\partial \hat{y}_2}{\partial h} \cdot \frac{\partial h}{\partial b} \]
RNN Paradigm: One to One (RNN unneeded)

No need for recurrent connections.

Classification: Frank or Froy

\[ Y_t \]

\[ X_t \] Image
RNN Paradigm: One to Many

Unrolled

Caption

Title

Diagram:

Image

Caption.

PowerPoint

Caption.
RNN Paradigm: Many to One
RNN Paradigm: Many to Many (Synced)

Frame-by-Frame Classifier/Regression

\[ h_0 \rightarrow [\text{RNN}] \rightarrow h_1 \rightarrow \ldots \rightarrow [\text{RNN}] \rightarrow h_o \]

Video Frame

\[ X_L \rightarrow Y_L \]
RNN Paradigm: Many to Many (Encoder/Decoder)
Parts-Of-Speech Example

You all must submit sentences for the dataset.

https://docs.google.com/spreadsheets/d/1HJmlehaYhGWclDo1t0k6i1VHxN15zr8ZmJ7Rf_VEal/edit#gid=1031300490

"I is a teeth"

Noun  Verb  Determiner  Noun

How do we pass this into a neural network?
Processing Natural Language with an NN

Here’s one way to convert text into numbers

1. Assign every word a unique number (e.g., 1 .. vocab_size)
2. Assign every part-of-speech a unique number (e.g., 1 .. num_classes)
3. Convert sentences into index tensors (using mapping from step 1)
4. Pass index tensors into an embedding layer (i.e., a simple lookup table)
5. Pass embedding outputs into the recurrent neural network (RNN)
6. Pass the RNN output into a fully-connected (FC) classification network
7. Convert the FC output into a part-of-speech (one-hot)
class POS_LSTM(torch.nn.Module):
    """Parts-of-speech LSTM model."""

    def __init__(self, vocab_size, embed_dim, hidden_dim, num_layers, parts_size):
        super().__init__()
        self.embed = torch.nn.Embedding(vocab_size, embed_dim)
        self.lstm = torch.nn.LSTM(embed_dim, hidden_dim, num_layers=num_layers)
        self.linear = torch.nn.Linear(hidden_dim, parts_size)

    def forward(self, X):
        X = self.embed(X)
        X, _ = self.lstm(X.unsqueeze(1))
        return self.linear(X)

        Output activation function handled by torch.nn.CrossEntropyLoss
RNNs

Advantages

• Process varying input length

• Model size remains constant

• Maintains historical information

Disadvantages

• Slower to computer

• Poor handling of long-term dependencies

• Does not consider future inputs to produce current state

• Largely replaced by transformers

Input: I love purple cats. Cats are neat.
Output: J'adore les chats violets. Les chats sont soignés.
We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely.

Attention Is All You Need (Vaswani et al, 2017)

Bahdanau et al., ICLR 2015

Summary

- Recurrent neural networks maintain an internal state (memory)
- This internal state is useful when data has a temporal component
- They were frequently used in translation and audio processing
- We don’t see them as much over the last few years, but the concepts are still worthwhile to know