Chapter 15 Report: Modeling Sequential Data Using Recurrent Neural Networks

Quan Pham

June 21, 2025

Contents

1 Abstract

2 Key concepts

2.1 Sequential data

Sequential data, also known as sequence data or sequences, is a type of data that its elements appear in a certain order, with interdependent relationships over time or position. The order of these elements is crucial, as it provide context and structural information that cannot be ignore during analysis. Changing the order can alter or even eliminate the meaning of the entire dataset.

Sequential data can have different lengths, from short sequences to very long ones. These are some typical examples: **Time series:** Daily stock price, sensor data (temperature, humidity), heart rate. **Natural language:** Words in a sentence, sentences in a paragraph. **Audio:** Voice waveforms, music sequences. **Video:** Sequences of frames **Biological sequence:** DNA sequences, protein sequences.

2.2 Categories of sequence modeling

If either the input or output is a sequence, the modeling task likely falls into one of these categories:

- Many-to-one: The input data is a sequence, but the output is a fixed-size vector or scalar, not a sequence.
- One-to-many: The input data is in standard format and not a sequence, but the output is a sequence.
- Many-to-many: Both the input and output arrays are sequences. This category can be further divided based on whether the input and output are synchronized.

2.3 Recurrent Neural Networks

A recurrent neural network (RNN) is any network that contains a cycle within its network connections, meaning that the value of some unit is directly, or indirectly, dependent on its own earlier outputs as an input

Computing activation in an RNN is very similar to standard multilayer peerceptrons and other types of feedforward NNs. For the hidden layer, the net input (preactivation), is computed as follows:

$$\boldsymbol{z}_h^{(t)} = \boldsymbol{W}_{xh}\boldsymbol{x}^{(t)} + \boldsymbol{W}_{hh}\boldsymbol{h}^{(t-1)} + \boldsymbol{b}_h$$

Then, the activations of the hidden units at the time step, t, are calculated as follows:

$$\boldsymbol{h}^{(t)} = \sigma_h(\boldsymbol{z}_h^{(t)}) = \sigma_h(\boldsymbol{W}_{xh}\boldsymbol{x}^{(t)} + \boldsymbol{W}_{hh}\boldsymbol{h}^{(t-1)} + \boldsymbol{b}_h)$$

Once the activations of the hidden units at the current time step are computed, then the activations of the hidden units at the time step are computed, then the activations of the

output units will be computed as follows:

$$\boldsymbol{o}^{(t)} = \sigma_o(\boldsymbol{W}_{ho}\boldsymbol{h}^{(t)} + \boldsymbol{b}_o)$$

2.4 Challenges of learning long-range interactions

Recurrent neural networks (RNNs) are theoretically capable of capturing dependencies across arbitrary sequence lengths. However, in practice, learning long-range interactions is challenging due to issues such as vanishing and exploding gradients during training. When back-propagating errors through many time steps, gradients can shrink exponentially (vanishing) or grow uncontrollably (exploding), making it difficult for the network to learn dependencies that span long intervals.

The vanishing gradient problem causes the network to "forget" information from earlier time steps, limiting its ability to model long-term dependencies. Conversely, exploding gradients can lead to unstable updates and numerical issues.

3 Algorithms and Models Studied

3.1 Converting raw text into sequence data

- remove HTML markups, punctuation and other non-letter characters
- extract emoticons
- split text into words
- encoding each unique token into intergers
- padding sequences for each batch

3.2 Backpropagation Through Time

$$L = \sum_{t=1}^{T} L^{(t)}$$

$$\frac{\partial L^{(t)}}{\partial \boldsymbol{W}_{hh}} = \frac{\partial L^{(t)}}{\partial \boldsymbol{o}^{(t)}} \times \frac{\partial \boldsymbol{o}^{(t)}}{\partial \boldsymbol{h}^{(t)}} \times (\sum_{k=1}^{t} \frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{h}^{(k)}} \times \frac{\partial \boldsymbol{h}^{(k)}}{\partial \boldsymbol{W}_{hh}})$$

$$\frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{h}^{(t)}} = \prod_{i=k+1}^{t} \frac{\partial \boldsymbol{h}^{(i)}}{\partial \boldsymbol{h}^{(i-1)}}$$

3.3 Truncated Backpropagation Through Time

Using gradient clipping, we specify a cut-off or threshold value for the gradients, and we assign this cut-off value to gradient values that exceed this value. In contrast, TBPTT simply limits the number of time steps that the signal can backpropagate after each forward pass. For example, even if the sequence has 100 elements or steps, we may only backpropagate the most recent 20 time steps.

3.4 Long-short Term Memory

LSTMs resemble standard recurrent neural networks but here each ordinary recurrent node is replaced by a memory cell. Each memory cell contains an internal state, i.e., a node with a self-connected recurrent edge of fixed weight 1, ensuring that the gradient can pass across many time steps without vanishing or exploding.

3.5 Gated Recurrent Units

GRUs have a simpler architecture than LSTMs; therefore, they are computationally more efficient, while their performance in some tasks, such as polyphonic music modeling, is comparable to LSTMs.

3.6 Bidirectional RNNs

4 Follow tutorial code in the book

4.1 Project one - predicting the sentiment of IMDb reviews

4.1.1 IMDb reviews dataset

The tutorial code loads the IMDb reviews datset using `torchtext` package. However, `torchtext` uses `torchdata` to load data, which doesn't support new python versions. I have to download the dataset from hugging face using `readparquet`.

```
splits = {'train': 'plain_text/train-00000-of-00001.parquet', 'test':
↪  'plain_text/test-00000-of-00001.parquet', 'unsupervised':
↪  'plain_text/unsupervised-00000-of-00001.parquet'}
df = pd.read_parquet("hf://datasets/stanfordnlp/imdb/" + splits["train"])

test_dataset = pd.read_parquet("hf://datasets/stanfordnlp/imdb/" +
↪  splits["test"])
```

Split train dataset to train set and validation set.

```
from sklearn.model_selection import train_test_split
train_dataset, valid_dataset = train_test_split(df, train_size=20000,
↪  test_size=5000, random_state=1)
```

4.1.2 Processing the text data

Tokenization

```
token_counts = Counter()

def tokenizer(text):
    text = re.sub(r'<[^>]*>', '', text)
    emoticons = re.findall(r'(?::|;|=)(?:-)?(?:\)|\(|D|P)', text.lower())
    text = re.sub(r'[\W]+', ' ', text.lower()) +\
        ' '.join(emoticons).replace('-', '')
    tokenized = text.split()
    return tokenized

for i, (text, label) in train_dataset.iterrows():
    tokens = tokenizer(text)
    token_counts.update(tokens)

print('Vocab-size:', len(token_counts))
```

```
Vocab-size: 69353
```

Vocab

```
from torchtext.vocab import vocab
sorted_by_freq_tuples = sorted(token_counts.items(), key=lambda x: x[1],
 ↪  reverse=True)
ordered_dict = OrderedDict(sorted_by_freq_tuples)
vocab = vocab(ordered_dict)
vocab.insert_token("<pad>", 0)
vocab.insert_token("<unk>", 1)
vocab.set_default_index(1)

print([vocab[token] for token in ['this', 'is', 'an', 'example']])
```

collate batch function.

```
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

def collate_batch(batch):
    label_list, text_list, lengths = [], [], []
    for _text, _label in batch:
        label_list.append(label_pipeine(_label))
        processed_text = torch.tensor(text_pipeline(_text),
         ↪  dtype=torch.int64)
        text_list.append(processed_text)
```

```
        lengths.append(processed_text.size(0))
    label_list = torch.tensor(label_list)
    lengths = torch.tensor(lengths)
    padded_text_list = nn.utils.rnn.pad_sequence(text_list,
    ↪  batch_first=True)
    return padded_text_list.to(device), label_list.to(device),
    ↪  lengths.to(device)
```

4.1.3 Loading data

Define TabularDataset class, which can return item from dataframe.

```
class TabularDataset(Dataset):
    def __init__(self, dataframe):
        self.dataframe = dataframe

    def __len__(self):
        return len(self.dataframe)

    def __getitem__(self, idx):
        return self.dataframe.iloc[idx]['text'],
        ↪  self.dataframe.iloc[idx]['label']
```

prepare data loaders.

```
batch_size = 32
train_set = TabularDataset(train_dataset)
valid_set = TabularDataset(valid_dataset)
test_set = TabularDataset(test_dataset)

train_dl = DataLoader(train_set, batch_size=batch_size, shuffle=True,
↪  collate_fn=collate_batch)
valid_dl = DataLoader(valid_set, batch_size=batch_size, shuffle=False,
↪  collate_fn=collate_batch)
test_dl = DataLoader(test_set, batch_size=batch_size, shuffle=False,
↪  collate_fn=collate_batch)
```

4.2 Project two - character-level language modeling with Pytorch

5 Pratical exploration with RNNs

6 Learnings and Challenges

6.1 Key Takeaways

6.2 Challenges Encountered

- it's difficult to find a unified definition for each concept.

6.3 Opening Questions

- embedding in character level models?

7 Conclusion