LSTM

Table of Contents

1. LSTM

1.1. Mathematical Formulation

When we train RNN, we usually face the problem of vanishing gradient or exploding gradient.

To solve this problem, we can use LSTM, whose architecture use simple addition and multiplication to update the memory.

LSTM introduces some new components compared to RNN:

Definition 1.1

- Memory Cell C_t: The memory cell is a vector that stores the long-term memory. It updates at every time step.
- Gate:
  - Input Gate i_t: Control what information should be added to the memory.
  - Forget Gate f_t: Control what information should be forgotten from the memory.
  - Output Gate o_t: Control what information should be output to the next memory cell C_{t+1}.
- Hidden State h_t: Just like RNN, LSTM has h_t, but the hidden state is a vector that stores the short-term memory.

Before introducing the forward process, we need to introduce the hadamard product.

Definition 1.2

The hadamard product of two vectors $a \in \mathbb{R}^n$ and $b \in \mathbb{R}^n$ is defined as:

$$a \odot b = [a_1 b_1, a_2 b_2, ..., a_n b_n]$$

For a LSTM of $[0, T]$ time steps, the whole forward process can be described as:

Definition 1.3

At timestep t:

1. Gate update:
   - Forget Gate: $f_t = \sigma\big(W_f \cdot [h_{t-1}, x_t] + b_f\big)$
   - Input Gate: $i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i), \tilde{C}_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_C)$

- Output Gate: $o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$

2. Memory Cell update: $C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t$
3. Hidden State update: $h_t = o_t \cdot \tanh(C_t)$
4. (Optional) Output: $y_t = f\left(W_y h_t + b_y\right)$

Let's do a **dimension analysis**:

Example 1.4

For an input sequence x of size $N_{\text{input}} = D$, the dimension of different parts in the LSTM at timestep t are:

- Gates:
  - $\sigma\left(\mathbb{R}^{N_{\text{hidden}} \times (N_{\text{input}} + N_{\text{hidden}})} \cdot \mathbb{R}^{(N_{\text{hidden}} + N_{\text{input}}) \times 1}\right) \to f_t \in \mathbb{R}^{N_{\text{hidden}} \times 1}$
  - Input Gate and Output Gate are similar
- Memory Cell:

  $C_t \in \left[\left(\mathbb{R}^{N_{\text{hidden}} \times 1} \odot \mathbb{R}^{N_{\text{hidden}} \times 1}\right) + \left(\mathbb{R}^{N_{\text{hidden}} \times 1} \odot \mathbb{R}^{N_{\text{hidden}} \times 1}\right)\right] \to C_t \in \mathbb{R}^{N_{\text{hidden}} \times 1}$

For the backward process, we also use **BPTT algorithm** to train the LSTM.

2. Code Implementation

Most of the code is similar to the RNN implementation, only the initialization of the model is different. Because we have different components.

```python
class MyLSTM(nn.Module):
    """A single layer LSTM model.

    Args:
        input_size (int): The size of the input features.
        hidden_size (int): The size of the hidden state.
        output_size (int): The size of the output features.
    """
    def __init__(self, input_size, hidden_size, output_size):
        super(MyLSTM, self).__init__()

        self.hidden_size = hidden_size
        self.output_size = output_size
        self.input_size = input_size

        # LSTM has memory cell, input gate, forget gate, and output gate
        self.input_gate = nn.Sequential(
            nn.Linear(input_size + hidden_size, hidden_size),
            nn.Sigmoid()
```

```
        )

        self.forget_gate = nn.Sequential(
            nn.Linear(input_size + hidden_size, hidden_size),
            nn.Sigmoid()
        )

        self.output_gate = nn.Sequential(
            nn.Linear(input_size + hidden_size, hidden_size),
            nn.Sigmoid()
        )

        self.candidate_cell_state = nn.Sequential(
            nn.Linear(input_size + hidden_size, hidden_size),
            nn.Tanh()
        )

        self.h2o = nn.Linear(hidden_size, output_size)
        self.cell_state = None
        self.hidden_state = None
```

And the forward process follows the mathematical formulation:

```python
def forward(self, x):
        batch_size, seq_len, input_size = x.size()
        hidden = torch.zeros(batch_size, self.hidden_size)
        cell_state = torch.zeros(batch_size, self.hidden_size)

        outputs = []

        for i in range(seq_len):
            x_t = x[:, i, :]
            # gate update
            f = self.forget_gate(torch.cat([x_t, hidden], dim=1))
            i = self.input_gate(torch.cat([x_t, hidden], dim=1))
            o = self.output_gate(torch.cat([x_t, hidden], dim=1))

            # cell state update
            candidate_cell_state =
            self.candidate_cell_state(torch.cat([x_t, hidden], dim=1))
            cell_state = f * cell_state + i * candidate_cell_state

            # hidden state update
```

```
            hidden = o * torch.tanh(cell_state)
            output = self.h2o(hidden)
            outputs.append(output.unsqueeze(1))

        self.cell_state = cell_state
        self.hidden_state = hidden
        return torch.cat(outputs, dim=1), self.hidden_state,
self.cell_state
```

3. Thinking

LSTM is a powerful model that can capture the long-term dependencies in the data. However, the composition of the model is quite complex and it introduces more parameters to train.